VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Shlukování pomocí algoritmu COBWEB

Clustering by COBWEB Algorithm

2010 Martin Vicher

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 7. května 2010 .....

Abstrakt

Tato diplomová práce se zabývá algoritmy pro shlukování dat, klasifikací těchto algoritmů a implementací několika vybraných shlukovacích algoritmů. U těchto vybraných algoritmů byla provedena optimalizace pro zpracování velkého počtu objektů velké dimenze. V této práci bylo zjištěno, že ne všechny popisované algoritmy jsou vhodné pro zpracování velkého počtu objeků velké dimenze. Na závěr byly experimentálně ověřeny vlastnosti těchto vybraných algoritmů.

Klíčová slova: Shlukování, COBWEB/CLASSIT, Farthest first traversial, K-means, DBscan, OPTICS, R-strom, KD-strom

Abstract

This diploma thesis is about clustering algorithms, their clasification and implementation of several choosen clustering algorithms. These choosen clustering algorithms were optimalized for processing high number of high dimensional objects. In this paper was found, that not all from these choosen clustering algorithms are suitable for processing high number of high dimensional objects. At the conclusion, properties of these chosen algorithms were experimentally verified.

Keywords: Clustering, COBWEB/CLASSIT, Farthest first traversial, K-means, DBscan, OPTICS, R-tree, KD-tree

Seznam použitých zkratek a symbolů

IDS	–	Intrusion detection system (Systémy pro detekci útoků)
FFT	–	Farthest first traversial
DBscan	–	Density based spatial clustering of applications with noise (Prostorové shlukování aplikací s šumem založené na hustotě)
OPTICS	–	Ordering points to identify the clustering structure (Uspořádání objektů pro určení struktury shluků)
MOO	–	Minimální ohraničující obdélník

Obsah

Seznam obrázků

Seznam tabulek

1 Úvod

V této diplomové práci se zabýváme shlukováním dat, což je metoda, která vytváří shluky objektů takovým způsobem, aby si objekty v jednom shluku byly velmi podobné a objekty v různých shlucích si byly podobné co nejméně. Jedná se o metodu takzvaného učení bez učitele. Se shlukováním dat se můžeme setkat například v oblasti strojového učení, dolování dat, analýze obrazu a bioinformatice [5, 6, 4].

V rámci této diplomové práce jsme provedli implementaci několika vybraných algoritmů (COBWEB/CLASSIT [8], K-means [10], Farthest first traversal [13], DBscan [7], OPTICS [2]) a pokusili se o jejich optimalizaci pro řídká data velké dimenze. Jsme si vědomi, že tyto zvolené shlukovací algoritmy představují jen malý zlomek existujících shlukovacích algoritmů. Dále uvádíme popis datových struktur, které mohou napomoci zefektivnit některé shlukovací algoritmy (v případě datové struktury KD-strom [20] byla provedena i její implementace). Jako základ pro tuto práci posloužila implementace shlukovacích algoritmů z nástroje Weka [25], avšak zde implementované algoritmy nepředpokládaly práci s velkým počtem řídkých dat velké dimenze. Tento nedostatek jsme se snažili odstranit jak optimalizací kódu (využitím předpokladu práce s řídkými daty), tak použitím jiné míry nepodobnosti (podobnosti) dvou vektorů (v případě algoritmu COBWEB/CLASSIT implementací jiného způsobu určení kategorizační utility). Tyto vybrané shlukovací algoritmy byly implementovány na platformě Microsoft .NET Framework verze 3.5 v jazyku C#. Nakonec jsme provedli experimenty s těmito vybranými slukovacími algoritmy na datech popisujících síťový provoz jak běžný, tak provoz představující síťový útok. Pomocí shlukovacích algoritmů jsme se snažili o správnou klasifikaci tohoto síťového provozu. Těmito experimenty jsme se snažili o ověření možnosti použití shlukovacích algoritmů v systémech pro detekci průniku (IDS).

Osnova práce

Tento práce je složena ze 7 kapitol. V první kapitola je úvod. Ve druhé kapitole definujeme shlukování (kapitola 2) a klasifikujeme jednotlivé shlukovací algoritmy podle způsobu vytváření shluků (podkapitola 2.1) a podle zppůsobu práce s daty (podkapitola 2.2).

Třetí kapitola popisuje jednotlivé vybrané shlukovací algoritmy (COBWEB/CLASSIT, K-means, Farthest first traversial, DBscan, OPTICS).

Ve čtvrté kapitole uvádíme datové struktury, které lze použít pro optimalizaci shlukovacích algoritmů. Datové struktury, které lze použít pro optimalizaci námi vybraných shlukovacích algoritmů jsou KD-strom (podkapitola 4.2) a R-strom(podkapitola 4.1).

V páté kapitole je uveden popis implementace (popis jednotlivých tříd a jejich rozdělení do knihoven).

Šestá kapitola popisuje experimenty, které byly s jednotlivými shlukovacími algoritmy provedeny a srovnání výsledků experimentů s výsledky experimentů v práci [23].

Výsledky této práce jsou shrnuty v závěru (kapitola 7), za kterým následuje seznam literatury použité pro tento text.

2 Klasifikace shlukovacích algoritmů

Shlukování dat může být definováno podle následující definice 2.1. Jedná se o metodu takzvaného učení bez učitele. Se shlukováním dat se můžeme setkat například v oblasti strojového učení, dolování dat, analýze obrazu a bioinformatice.

Definice 2.1 *Shlukování dat je metoda, která vytváří shluky objektů takovým způsobem, aby si objekty v jednom shluku byly velmi podobné a objekty v různých shlucích si byly podobné co nejmíň. Tedy, máme-li množinu objektů* $O = \{O_1, ..., O_n\}$ *a míru nepodobnosti objektů* v*, shlukem nazveme takovou podmnožinu* $X \subset O$*, pro kterou platí:*

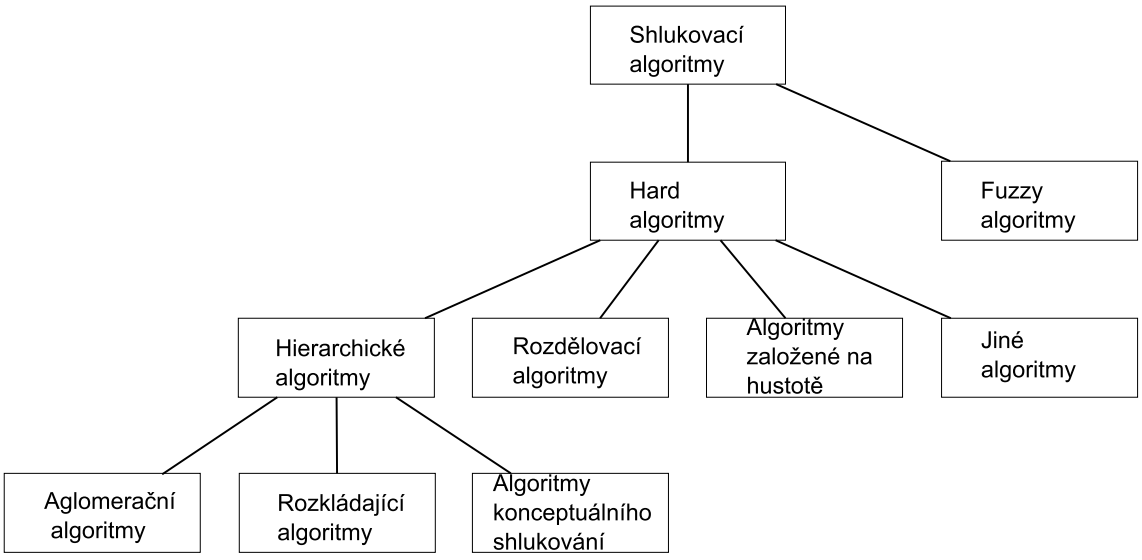
$$\max_{O_i \in X, O_j \in X}(v(O_i, O_j)) < \min_{O_i \in X, O_k \notin X}(v(O_k, O_i)).$$

Definice 2.1 byla převzata z [21].

Jednotivé shlukovací algoritmy můžeme rozdělit několika způsoby, například podle toho, jakým způsobem vytvářejí shluky objektů nebo podle toho, zda se jedná o algoritmy inkrementální či nikoliv.

2.1 Klasifikace podle způsobu vytváření shluků

Podle [4] můžeme algoritmy shlukování rozdělit do několika hlavních skupin (viz obrázek 1).



Obrázek 1: Rozdělení shlukovacích algoritmů

Obecně lze rozdělit shlukovací algoritmy na dva typy: hard a fuzzy algoritmy. První jmenovaný zařazuje objekty pouze do jednoho shluku, naproti tomu fuzzy algoritmy mohou objekt zařadit do více shluků, ale do každého pouze s určitou pravděpodobností. Fuzzy algoritmy nejsou pro tento text důležité a proto je zde nebudeme uvádět.

Hard algoritmy můžeme dále rozdělit na hierarchické algoritmy, rozdělovací algoritmy, algoritmy založené na hustotě a jiné algoritmy.

2.1.1 Hierachické algoritmy

Tyto algoritmy se dají rozdělit do tří skupin: na aglomerační, rozkládající a konceptuální. Rozkládající algoritmy zařadí objekty do jednoho shluku a ten poté dělí na menší shluky podle podobnosti objektů. Aglomerační postupují přesně naopak, kdy každý objekt patří do pouze jednoho shluku a tyto shluky se postupně slučují podle jejich podobnosti. Tímto vznikne hierarchická struktura shluků. Speciálním případem hierarchických algoritmů jsou algoritmy konceptuálního shlukování, kombinující aglomerační a rozkládající přístup.

2.1.2 Algoritmy konceptuálního shlukování

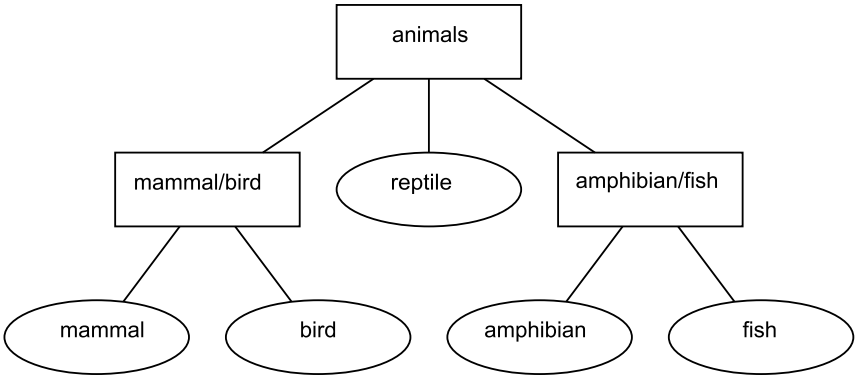
Algoritmy konceptuálního shlukování vytváří inkrementálním způsobem strukturu z dat dělením pozorovaných objektů do podtříd. Výsledkem těchto algoritmů je klasifikační strom. Každý uzel tohoto stromu obsahuje objekty svých podřízených uzlů, tedy kořen tohoto stromu obsahuje celou množinu objektů. Podle výše uvedených klasifikací se jedná o inkrementální hierarchický algoritmus, kombinující jak agregační, tak rozkládající přístup. Tabulka 1 a obrázek 2 uvádí příklad, jak může takový klasifikační strom vypadat. Existuje mnoho algoritmů konceptuálního shlukování, například CLUSTER/2 [19], CYRUS [15], COBWEB/CLASSIT. Algoritmus COBWEB/CLASSIT budeme popisovat podrobněji.

Name	BodyCover	HeartChamber	BodyTemp	Fertilization
mammal	hair	four	regulated	internal
bird	feathers	four	regulated	internal
reptile	cornified-skin	imperfect-four	unregulated	internal
amphibian	moist-skin	three	unregulated	external
fish	scales	two	unregulated	external

Tabulka 1: Příklad: popis zvířat [8]

2.1.3 Rozdělovací algoritmy

Rozdělovací algoritmy rozdělují objekty do několika disjunktních množin a tak vytvářejí jedinou úroveň nepřekrývajících se shluků. Je však problém určit, kolik shluků má algoritmus odhalit. Mezi tyto algoritmy patří například algoritmus K-means a Farthest first traversal.



Obrázek 2: Příklad: kategorizační strom pro tabulku 1 [8]

2.1.4 Algoritmy založené na hustotě

Algoritmy založené na hustotě se zakládají na principu, že shluk je množina propojených prvků se stejnou hustotou rozložení. Mezi jejich výhody patří, že jsou odolné vůči šumu v datech a výstředním hodnotám. Dokáží najít i shluky s nepravidelným tvarem. Mezi tyto algoritmy patří například algoritmus OPTICS, DBscan a jeho varianty.

2.1.5 Jiné algoritmy

Zde patří například algoritmy založené na mřížkách nebo algoritmy využívající neuronové sítě. Pro tento text nejsou důležité, a proto se jimi v této práci nebudeme zabývat.

2.2 Klasifikace podle způsobu práce s daty

Dalším pohledem na shlukovací algoritmy je, zda jsou **inkrementální** či **neinkrementální**. Mnoho algoritmů je neinkrementálních, tedy všechny objekty musí být přítomny hned na počátku spuštění algoritmu. V případě, že by se objevily nové objekty, není možné je nově zpracovat bez toho, aniž by byly opětovně zpracovány i předchozí objekty. Oproti tomu inkrementální systémy umožňují postupné zpracování objektů bez toho, aby se musely znovu zpracovat předchozí objekty. Hlavním důvodem pro vývoj inkrementálních algoritmů je, že v reálném světě neznáme všechny objekty pro zpracování hned při spuštění systému. V tomto scénáři (u neinkrementálního systému) by i při jednom novém objektu musela být znovu zpracována celá množina již jednou zpracovaných objektů.

3 Vybrané shlukovací algoritmy

V následujících několika podkapitolách se zaměříme na jednotlivé vybrané shlukovací algoritmy - COBWEB/CLASSIT, K-means, Farthest first traversial, DBscan a OPTICS. Tyto algoritmy jsme vybrali na základě jejich existence v nástroji Weka (z jehož implementace jsme vycházeli) a u tyto algoritmy jsme se snažili zoptimalizovat pro velký počet dat velké dimenze. U jednotlivých algoritmů je uveden princip, popis daného algoritmu a možnosti optimalizace daného algoritmu.

3.1 COBWEB/CLASSIT

Algoritmus COBWEB byl inspirován prací dvojice Michalski, Stepp [19] a algoritmy UNIMEM [17, 16], CYRUS. COBWEB je inkrementální algoritmus pro hierarchické konceptuální shlukování. Tento algoritmus provádí vyhledávání extrému přes prostor hierarchického klasifikačního schématu použitím operátorů, umožňujících obousměrné cestování přes tento prostor. V dalších částech se zaměříme na to, jakým způsobem je určena kategorizační utilita, jakým způsobem je tento koncept reprezentován, a nakonec se podíváme na jednotlivé operátory a řídící strukturu COBWEBu. Algoritmus CLASSIT je rozšířením algoritmu COBWEB pro zpracování objektů s numerickými atributy, algoritmus COBWEB zpracovává pouze kategoriální atributy. Rozdíl mezi oběma algoritmy je dán použitou kategorizační utilitou, další části algoritmu jsou pro COBWEB i CLASSIT stejné.

3.1.1 Reprezentace konceptu

Základ každého klasifikačního schématu je v reprezentaci individuálních konceptů (tříd). Reprezentace, kdy seznam hodnot atributu je spojen s jejich pravděpodobností, je pravděpodobnostní koncept (tabulka 2). U COBWEBu značí pravděpodobnostní koncept každý uzel v klasifikačním stromu a sčítá objekty klasifikované daným uzlem. U stromů pravděpodobnostního konceptu značkuje pravděpodobnostní deskriptor uzly stromu. Klasifikace, používající strom pravděpodobnostního konceptu, je vykonána použitím funkce částečného porovnání k sestupu stromem podél cesty nejlépe vyhovujících uzlů. Následující části ukazují, jak jsou tyto hlavní procedury (operátory) adaptovány algoritmem COBWEB pro aktualizaci stromu.

Attributes	Values and Probabilities
BodyCover	scales[0.33], moist-skin[0.33], hair[0.33]
HeartChamber	two[0.33], three[0.33], four[0.33]
BodyTemp	unregulated[0.67], regulated[0.33]
Fertilization	external[0.67], internal[0.33]

Tabulka 2: Příklad: pravděpodobnostní reprezentace (fish, amphibian, mammal) odvozená z tabulky 1 [8]

3.1.2 Kategorizační utilita

COBWEB používá k řízení vyhledávání heuristickou míru, nazývanou kategorizační utilita. Níže popisovaná metrika ve vzorci (3) slouží pro zpracovávání kategoriálních atributů. Tato metrika byla vytvořena dvojicí Gluck a Corter původně jako průměrná předpověď základní úrovně v lidské klasifikační hierarchii [11]. Na kategorizační utilitu může být pohlíženo jako na funkci, která ohodnocuje tradiční hodnoty shlukování (tedy podobnost objektů uvnitř stejné třídy a nepodobnost objektů různých tříd). Na kategorizační utilitu se můžeme dívat jako na kompromis mezi podobností uvnitř třídy a mezitřídní nepodobností objektů, kde objekty chápeme jako pár atribut-hodnota jako v tabulce 1. Podobnost uvnitř třídy je zachycena podmíněnou pravděpodobností $P(A_i = V_{ij}|C_k)$, kde $A_i = V_{ij}$ je pár atribut-hodnota a C_k je třída. Čím větší je pravděpodobnost, tím větší část členů třídy sdílí hodnotu, a tak je tato hodnota u členů třídy očekávanější. Mezitřídní nepodobnost je vyjádřena vztahem $P(C_k|A_i = V_{ij})$ vyjadřujícím, že čím větší pravděpodobnost, tím méně objektů v různých třídách sdílí tutéž hodnotu. Vzorec:

$$\sum_{k=1}^{n} \sum_{i} \sum_{j} P(A_i = V_{ij}) P(C_k|A_i = V_{ij}) P(A_i = V_{ij}|C_k) \tag{1}$$

představuje kompromis mezi podobností uvnitř třídy a mezitřídní nepodobností. Úpravou vzorce (1) (použitím Bayesova pravidla) dostaneme vzorec:

$$\sum_{k=1}^{n} P(C_k) \sum_{i} \sum_{j} P(A_i = V_{ij}|C_k)^2. \tag{2}$$

Vzorec $\sum_i \sum_j P(A_i = V_{ij}|C_k)^2$ udává očekávaný počet hodnot atributu, které mohou být správně uhádnuty pro libovolného člena třídy C_k. Nakonec Gluck a Corter [11] definovali kategorizační utilitu jako zvýšení očekávaného počtu hodnot atributu, které mohou být správně uhádnuty ($P(C_k) \sum_i \sum_j P(A_i = V_{ij}|C_k)^2$) určeným rozdělením ($(C_1 ... C_n)$) mimo očekávaný počet správných uhádnutí bez této znalosti ($\sum_i \sum_j P(A_i = V_{ij})^2$). Vzorec pak vypadá takto:

$$CU_p = \frac{\sum_{k=1}^{n} P(C_k)[\sum_i \sum_j P(A_i = V_{ij}|C_k)^2 - \sum_i \sum_j P(A_i = V_{ij})^2]}{n}. \tag{3}$$

Dělitel představuje počet kategorií v rozdělení. Toto zprůměrování umožňuje porovnání rozdělení různých velikostí. Více informací o kategorizační utilitě můžete najít například v [8].

Pro zpracování numerických atributů se využívá upravené kategorizační utility (4). Algoritmus využívající této utility je znám jako CLASSIT. Tato utilita je dána následujícím vztahem:

$$CU_p = \frac{\sum_k P(C_k) \sum_i (\frac{1}{\sigma_{i,k}} - \frac{1}{\sigma_{i,p}})}{K}, \tag{4}$$

kde $\sigma_{i,p}$ a $\sigma_{i,k}$ znamenají:

- $\sigma_{i,p}$ - standardní odchylka atributu i v rodičovském uzlu,
- $\sigma_{i,k}$ - standardní odchylka atributu i v uzlu potomka.

Vzorec (4) je vhodný pro zpracování objetů jejichž atributy mají normální rozdělení. V případě Katzova rozdělení, vhodného pro zpracování textových dat, je daná kategorizační utilita dána následujícím vztahem:

$$CU_p = \frac{1}{K} \sum_k P(C_k)[\sum_i CU_{i,k} - \sum_i CU_{i,p}], \tag{5}$$

kde $CU_{i,k}$ a $CU_{i,p}$ jsou dány následujícím vztahem:

$$CU_{i,k} = 1 - \frac{2 * df(N - \frac{cf * df}{2 * cf - df})}{N^2}, \tag{6}$$

kde jednotlivé symboly znamenají:

- N - počet dokumentů v kolekci,
- K - počet potomků uzlu klasifikačního stromu, pro který počítáme kategorizační utilitu,
- df - frekvence dokumentu - počet dokumentů v celé kolekci, které obsahují dané slovo,
- cf - frekvence kolekce - udává, kolikrát se dané slovo objeví v kolekci dokumentů, je získána sčítáním výskytu daného slova v jednotlivých dokumentech kolekce.

Pro jiné statistické rozdělení atributů, například Negativně binomického či Poissonova, je možné použít upravenou kategorizační utilitu, ale její výpočetní složitost je velká a nehodí se pro použití. Pro více informací o způsobu určení kategorizačních utilit doporučujeme článek [22].

3.1.3 Operátory

COBWEB/CLASSIT inkrementálně zařazuje objekty do klasifikačního stromu, kde každý uzel představuje pravděpodobnostní koncept reprezentující třídu objektu. Zařazení objektu je procesem klasifikace objektu sestupujícího stromem podél příslušné cesty, aktualizace čítačů podél cesty a vykonání jednoho z možných operátorů v každé úrovni. Existují tyto čtyři operátory:

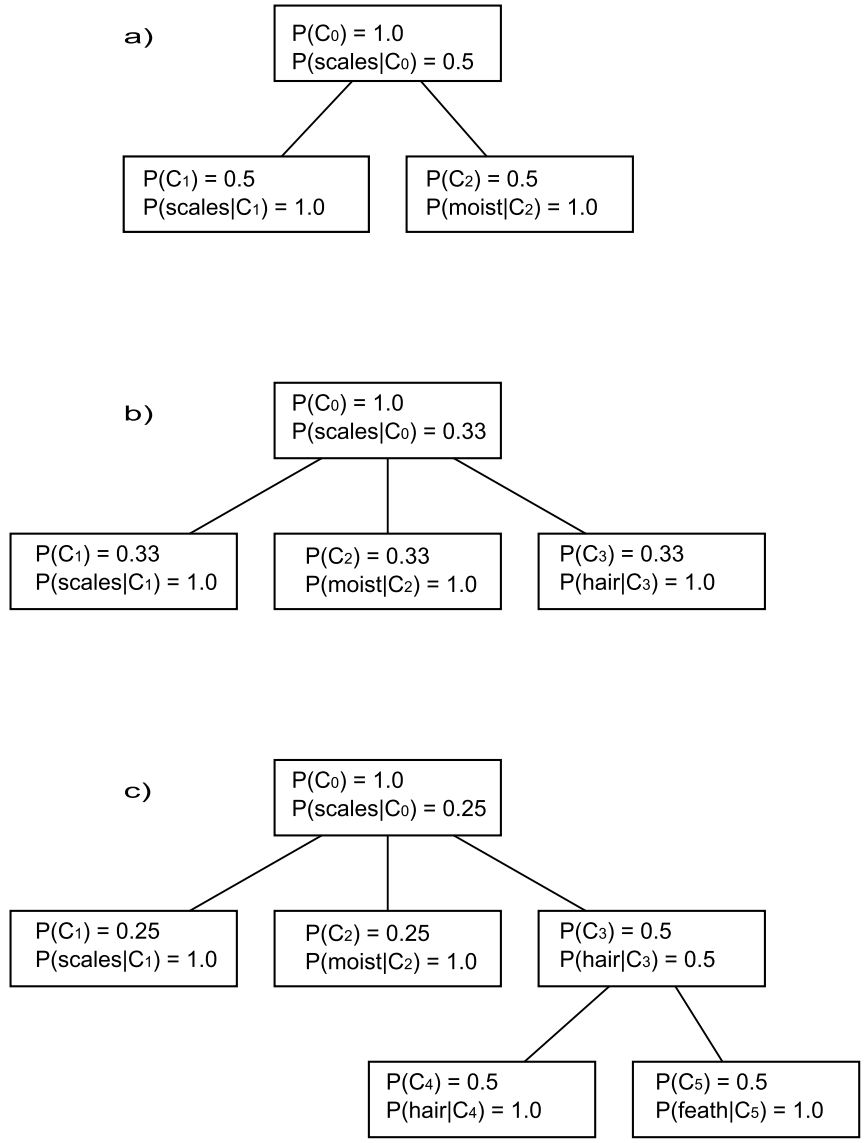
1. vložení objektu do existující třídy,
2. vytvoření nové třídy,
3. spojení dvou tříd do jediné třídy,
4. rozdělení jedné třídy na několik tříd.

Operátor vložení objektu do existující třídy.
První možností aktualizace množiny tříd je jednoduše přidat objekt do již existující třídy. Když COBWEB/CLASSIT zjišťuje, která kategorie by byla nejlepší pro umístění nového objektu, COBWEB/CLASSIT pokusně vkládá objekt do každé kategorie. Rozdělení, která vyplývají z přidání nového objektu do existujícího uzlu, jsou ohodnoceny kategorizační utilitou dle vzorce (3). Uzel, který má za následek nejlepší rozdělení, je pak vyhodnocen jako nejlepší pro vložení nového objektu.

Operátor vytvoření nové třídy.
Mimo možnosti vložení objektu do existující třídy umožňuje COBWEB/CLASSIT vytvořit novou třídu. Kvalita rozdělení, která vyplývá z umístění objektu do nejlepšího existujícího uzlu, je porovnána s rozdělením, vzniklým vytvořením nové třídy obsahující pouze nový objekt. V závislosti na tom které rozdělení je lepší, bude objekt buď umístěn do nejlepší existující třídy nebo do nově vytvořené třídy. Tento operátor umožňuje algoritmu COBWEB/CLASSIT zvyšovat počet tříd v rozdělení.

Příklad použití prvních dvou operátorů
Obrázek 3, převzatý z [8], ukazuje použití dvou již představených operátorů. Obrázek 3 a) ukazuje předtvořený klasifikační strom z objektů fish a amphibian uvedených v tabulce 1. Každý uzel (třída) stromu zobrazuje pravděpodobnosti třídy a pravděpodobnost hodnoty atributu podmíněnou na členství v třídě. Z důvodů nedostatku místa zobrazují uzly stromu pouze podmíněné pravděpodobnosti pro jednu hodnotu atributu, například uzel C_0 na obrázku 3 b) je specifikovaný pravděpodobnostním konceptem zobrazeným v tabulce 2. Obrázek 3 b) ukazuje vytvoření nové třídy. Strom na obrázku 3 a) se přetvoří v b) zařazením objektu mammal z tabulky 1. Vytvoření nové třídy má lepší výsledky (podle kategorizační utility) než zařazení objektu do již existující třídy. Obrázek 3 c) znázorňuje přidání objektu do již existující třídy. Přidání objektu bird do třídy mammal má za následek opět nejlepší možné rozdělení. Přidáním objektu bird do stromu na obrázku 3 b) způsobí přepočítání pravděpodobností a dále (jelikož je třída C_3 listem

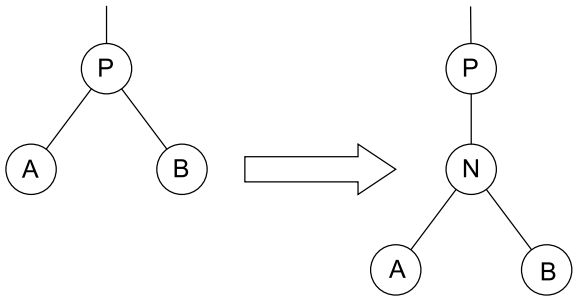
stromu) dojde k vytvoření dvou jeho potomků (jeden pro nový objekt a druhý pro dříve klasifikovaný).



Obrázek 3: Příklad: přidání dvou objektů mammal a bird do existujícího klasifikačního stromu. Každý uzel představuje objekt třídy C_i, který je zachycen svou množinou pravděpodobností [8]

Operátor spojení a rozštěpení.
První dva operátory nejsou vždy nejlepší cestou, protože jsou velice náchylné na seřazení vstupních dat. Algoritmus COBWEB/CLASSIT pro zajištění nezávislosti na počátečním pořadí dat zavádí další dva operátory. Jsou to operátory spojení a rozštěpení. Operátor spojení vezme dva uzly stejné úrovně a zkombinuje je v naději, že výsledné rozdělení bude lepší. Způsob, jakým je spojení provedeno, je následující:

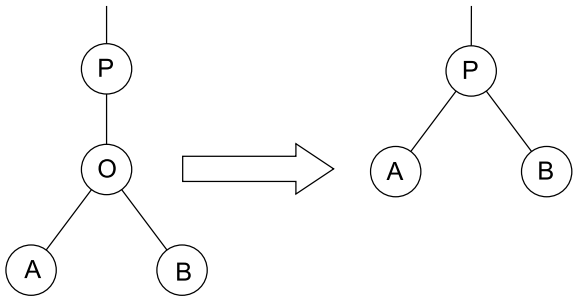
1. Vytvoření nového uzlu a sečtení počtu párů atribut-hodnota uzlů, které se spojují.
2. Dva spojované uzly jsou připojeny k nově vytvořenému uzlu jako jeho potomci.



Obrázek 4: Příklad: aplikace operátoru spojování uzlů [8].

Ačkoliv může dojít ke spojování uzlů u všech možných párů uzlů kdykoliv je objekt pozorován, není to žádoucí, jelikož je tato operace nákladná. Místo toho dojde ke spojování uzlů pouze u dvou nejlepších uzlů (určených kategorizační utilitou).

Stejně jako operátor spojení může i operátor rozštěpení zvýšit kvalitu rozdělení. Uzel rozdělení může být smazán a jeho potomci mohou být povýšeni. Rozštěpení může nastat pouze u potomků nejlepšího hostitele mezi všemi kategoriemi. Operace spojování uzlů a



Obrázek 5: Příklad: aplikace operátoru rozštěpení uzlu [8].

rozštěpení uzlu jsou navzájem inverzí operace. Jedna může být provedena pro odstranění následku druhé, což je patrné i z obrázků 4 a 5.

3.1.4 Popis algoritmu COBWEB/CLASSIT.

Algoritmus 1 představuje zjednodušenou řídící strukturu, kterou COBWEB/CLASSIT používá k organizaci operátorů. Použitím této strategie má systém sklon konvergovat na klasifikační strom, ve kterém první úroveň (kořen je nultá úroveň) je optimálním rozdělením (podle kategorizační utility) celé množiny objektů.

Algoritmus 1 : Algoritmus COBWEB/CLASSIT [8]

```
1: function COBWEB/CLASSIT(Objekt o, uzel_klasifikačního_stromu u)
2:     aktualizace čítače kořene
3:     if (u je list) then
4:         RETURN rozšířený list k uložení nového objektu
5:     else
6:         Najdi potomka uzlu u, který bude nejlépe hostovat o a vykonej jeden z následujících kroků:
7:         v případě vhodnosti vytvoření nového uzlu,
8:         v případě vhodnosti spojení dvou uzlů a volání COBWEB/CLASSIT(o, u),
9:         v případě vhodnosti rozdělení uzlu a volání COBWEB/CLASSIT(o, u),
10:        if (nic z výše uvedeného nebylo provedeno) then
11:            volaní COBWEB/CLASSIT(o, nejlepší potomek uzlu u).
12:        end if
13:    end if
14: end function
```

Tento algoritmus používá ještě dva parametry důležité pro kategorizační utilitu. A jsou to Acuity a Cutoff. První jmenovaný udává minimální hodnotu směrodatné odchylky a druhým určuje hraniční hodnotu kategorizační utility pro vytvoření nového uzlu (bude-li kategorizační utilita menší než Cutoff, nebude vytvořen nový uzel a objekt bude přidán do existujícího uzlu). Tyto parametry nastavuje uživatel algoritmu a pro jejich určení neexistuje jiná metodika než je stanovit na základě experimentů s konkrétními daty.

3.1.5 Možnosti optimalizace algoritmu

Při optimalizaci tohoto algoritmu pro řídká data, na rozdíl od hustých dat, kdy víme, že všechny objekty (vektory) mají všechny atributy a nemusí se existence jednotlivých atributů při výpočtu kategorizační utility testovat, je nutné toto testování atributů provádět. To má za následek znatelné snížení výkonu tohoto algoritmu. Toto testování je nutné provádět u obou implementovaných algoritmů pro výpočet kategorizační utility.

3.2 Farthest first traversal

Algoritmus Farthest first traversial (FFT) byl navržen pro řešení problému K-center [13]. Jedná se o rozdělovací shlukovací algoritmus. Tento algoritmus nejprve vybere K objektů jako středy shluků a následně přiřadí ostatní objekty do příslušných shluků (podle míry nepodobnosti ke středům shluků). První střed shluku je vybrán náhodně, druhý jako nejméně podobný prvnímu a každý další jako ten, jehož nejmenší hodnota míry nepodobnosti k již zvoleným středům shluků je největší. Jinými slovy je nejméně podobný objekt x z množiny objektů S definován jako $max_x\{min(x, j), j \in S\}$ [24].

3.2.1 Popis algoritmu FFT

Časová složitost algoritmu je $\Theta(Nk)$, kde N je počet shlukovaných objektů a k je počet generovaných shluků. Algoritmus FFT je popsán následujícím algoritmem 2.

Algoritmus 2 : Představuje algoritmus FFT [24]

1: **function** FFT(sada_objektů N, počet_shluků k)
2: náhodná volba prvního středu shluku
3: **for** ($i = 0$;$i < k$;$i++$) **do**
4: **for all** ($o \in N$) **do**
5: **if** (o není středem shluku) **then**
6: je vypočtena míra nepodobnosti mezi o a jednotlivými středy shluku a pro každý o je uložena nejmenší takto vypočtená hodnota míry nepodobnosti
7: **end if**
8: **end for**
9: jako nový střed shluku je vybrán ten objekt o, jehož nejmenší hodnota míry nepodobnosti ke středům shluků je maximální
10: **end for**
11: **for all** ($o \in N$) **do**
12: **if** (o není středem shluku) **then**
13: je vypočtena jeho míra nepodobnosti k jednotlivým středům shluků a je přidán do toho shluku, jehož středu je nejpodobnější
14: **end if**
15: **end for**
16: **end function**

Problémem při použití tohoto algoritmu je způsob určení, kolik shluků má algoritmus FFT vygenerovat. Způsob, jakým je vypočtena míra nepodobnosti mezi dvěmi objekty, je popsán v následující podkapitole.

3.2.2 Míry podobnosti a nepodobnosti objektů

Pro shlukování je důležitý pojem podobnosti (nepodobnosti) objektů. Některé shlukovací algoritmy při své činnosti určují míru podobnostin (nepodobnosti) dvou objektů. Pro míry podobnosti platí, že čím jsou si dva objekty bližší, tím je pro ně tato míra větší. Při porovnání dvou totožných objektů nastává problém jak velkou hodnotu má tato míra přiřadit. Tento problém řeší míry nepodobnosti dvou objektů, které jsou pro dva totožné objekty rovny 0 a čím víc se budou tyto dva objekty od sebe lišit tím větší pro ně tato míra bude. V této sekci se zaměříme na míry podobnosti a nepodobnosti objektů implementované v rámci této práce. Jedná se o následující dvě míry nepodobnosti:

- Euklidovská míra,
- Manhattenská míra

a jednu míru podobnosti:

- Kosinova míra.

Euklidovská míra

Euklidovská míra je nejčastěji používanou mírou nepodobnosti dvou objektů. Pro dva objekty x a y o d atributech se Euklidovská míra vypočte podle vzorce:

$$D_{euc}(x,y) = [\sum_{j=1}^{d}(x_j - y_j)^2]^{\frac{1}{2}} = [(x-y)(x-y)^T]^{\frac{1}{2}}, \tag{7}$$

kde:

- x_j a y_j jsou hodnoty j-tých atributů objektů x a y.

Pro použití této míry je třeba nejprve jednotlivé hodnoty atributů normalizovat. Normalizace je provedena tak, aby jednotlivé atributy nabývaly hodnot z intervalu od 0 do 1. Jednotlivé vektory se normalizují pomocí vzorce:

$$a_i = \frac{v_i - minv_i}{maxv_i - minv_i}, \tag{8}$$

kde:

- a_i je normalizovaná hodnota i-tého atributu,
- v_i je hodnota i-tého atributu před normalizací,
- $minv_i$ je minimální hodnota i-tého atributu ze všech objektů datové sady,
- $maxv_i$ je maximální hodnota i-tého atributu ze všech objektů datové sady.

Jednotlivé vzorce jsou převzaty z [10].

Manhattenská míra

Manhattenská míra je také nazývána vzdáleností městských bloků a je definována jako součet vzdáleností všech atributů. Jedná se o míru nepodobnosti dvou objektů. Pro dva objekty x a y o d atributech se Manhattenská míra vypočte podle vzorce:

$$D_{man}(x, y) = \sum_{j=1}^{d} |x_j - y_j|, \tag{9}$$

kde:

- x_j a y_j jsou hodnoty j-tých atributů objektů x a y.

V případě, že objektům x a y chybí hodnota některých atributů, lze použít upravený vzorec:

$$D_{manw}(x, y) = \sum_{j=1}^{d} \frac{w_j |x_j - y_j|}{\sum_{i=1}^{d} w_i}, \tag{10}$$

kde:

- x_j a y_j jsou hodnoty j-tých atributů objektů x a y,
- w_j nabývá hodnoty 0, jestliže hodnoty j-tých atributů objektů x a y chybí, jinak nabývá hodnotu 1.

Jednotlivé vzorce jsou převzaty z [10].

Kosinova míra

Vzorcem (12) určíme míru podobnosti dvou objektů o d atributech. Tato míra podobnosti nabývá hodnot z intervalu od 0 do 1. Chceme-li ji použít ve shlukovacích algoritmech je nutné ji převést na míru nepodobnosti dvou objektů dle vzorce:

$$D_{cos}(x, y) = 1 - coss(x, y), \tag{11}$$

kde $coss(x, y)$ je určeno vztahem:

$$coss(x, y) = \frac{\sum_{d}^{j=1} (x_j y_j)}{\sqrt{\sum_{j=1}^{d} (x_j)^2 \sum_{j=1}^{d} (y_j)^2}}, \tag{12}$$

kde:

- x_j a y_j jsou hodnoty j-tých atributů objektů x a y.

Jednotlivé vzorce jsou převzaty z [10].

3.2.3 Možnosti optimalizace algoritmu

Optimalizace tohoto algoritmu pro řídká data spočívala v úpravě jednotlivých funkcí pro výpočet měr nepodobností dvou objektů tak, aby nemuselo docházet k procházení všech atributů jednotlivých objektů, ale pouze těch nenulových. Už z porovnání vzorců jednotlivých měr se jeví jako nejlepší použití Kosinovy míry (v případě tohoto algoritmu se při experimentech neprojevila výhoda této míry - viz kapitola 6).

3.3 K-means

Algoritmus K-means je podle výše zmíněné klasifikace rozdělovacím shlukovacím algoritmem. Jedná se o jeden z nejjednodušších algoritmů učení bez učitele. Algoritmus jednoduchým způsobem klasifikuje danou množinu dat na fixní, předem daný počet (K) shluků. Hlavní myšlenkou celého algoritmu je nalezení K středů (pro každý shluk jeden střed) shluků. Otázkou je, jak umístit tyto středy shluků, jelikož výsledky tohoto výběru značně ovlivní výsledné shluky. Nejlepší by bylo vybrat středy shluků sobě co nejméně podobné. Dalším krokem je přiřadit každý objekt datové sady ke středu shluku, jemuž je nejpodobnější. Po ukončení tohoto zařazení nastane další krok a to určení (K) nových středů jednotlivých shluků (středy jsou odvozeny z objektů jednotlivých shluků). Opět se provede znovuzařazení objektů do jednotlivých shluků podle jejich míry nepodobnosti k novým středům shluků. Tyto kroky se opakují dokud nezjistíme, že se tyto středy již dále nemění.

3.3.1 Popis algoritmu K-means

Časová složitost algoritmu je $\Theta(Nkt)$, kde N je počet shlukovaných objektů, k je počet generovaných shluků a t je počet iterací. Algoritmus K-means je popsán následujícím algoritmem 3.

Algoritmus 3 : Představuje algoritmus K-means [10]

1: **function** KMEANS(sada_objektů N, počet_shluků k, počet_iterací t)
2: náhodný výběr k objektů z N jako středů shluků
3: **repeat**
4: **for all** ($o \in N$) **do**
5: je vypočtena míra nepodobnosti mezi objektem o a jednotlivými středy shluků
6: objekt o je přidán do toho shluku, k jehož středu je nejblíže
7: **end for**
8: výpočet nových středů shluků
9: **until** (není splněna podmínka ukončení (středy shluků se již moc nemění nebo je dosaženo maxima iterací))
10: **end function**

Algoritmus používá pro výpočet míry nepodobnosti dvou objektů jednu z námi implementovaných měr (Euklidova míra, Manhattenská míra a Kosinova míra).

Tento algoritmus je velmi závislý na náhodné počáteční volbě středů shluků (místo náhodné volby je možno použít středy shluků produkované z výše uvedeného algoritmu Farthest first traversal). Vliv počáteční náhodné inicializace lze částečně odstranit několikanásobným spuštěním algoritmu, kdy se vyzkouší různé počáteční inicializace středů shluků [10, 1, 14]. Je pak na uživateli, aby si vybral, kdy algoritmus nalezl správné rozdělení shluků. Dalším problémem při použití tohoto algoritmu je, jakým způsobem stanovit kolik shluků má algoritmus vygenerovat.

3.3.2 Možnosti optimalizace algoritmu

Optimalizace tohoto algoritmu pro řídká data spočívala v úpravě jednotlivých funkcí pro výpočet míry nepodobnosti dvou objekty tak, aby nemuselo docházet k procházení všech atributů jednotlivých objektů, ale pouze těch nenulových. Z tohoto důvodu se jeví jako nejlepší použití Kosinovy míry (viz kapitola 6).

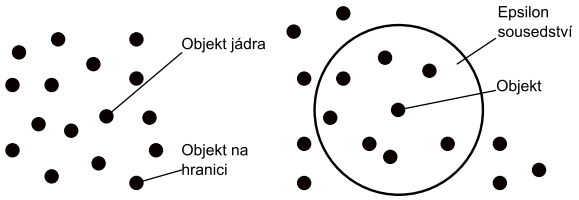
3.4 DBscan

Tento algoritmus patří mezi algoritmy založené na hustotě. Je jedním z nejrozšířenějších shlukovacích algoritmů. Na jednotlivé shluky pohlíží jako na oblasti s velkou hustotou objektů, které jsou odděleny oblastmi s nízkou hustotou objektů. Před popisem samotného algoritmu je třeba uvést několik pojmů.

Definice 3.1 *ϵ-sousedství objektu x je definováno jako:*

$$N_\epsilon(x) = y \in D : d(x, y) \leq \epsilon$$

kde D, je vstupní množina objektů, ϵ je zadaná velikost ϵ-sousedství a d je funkce pro určení míry nepodobnosti dvou objektů (touto mírou může být opět Euklidova míra, Manhattenská míra, Kosinova míra podobnosti převedená na míru nepodobnosti dle vzorce (11)).



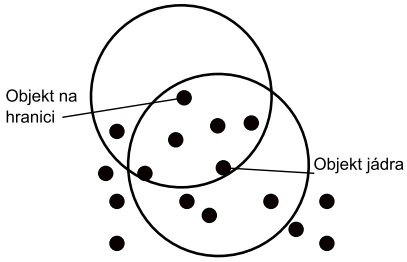
Obrázek 6: Příklad: ϵ-sousedství objektu ve dvourozměrném prostoru

Definice 3.2 *Přímo hustotou dosažitelné (Directly density reachability)*
Objekt x je přímo hustotou dosažitelný z objektu y, když

1. $x \in N_\epsilon(y)$
2. $|N_\epsilon(y)| \geq N_{min}$, *kde* $|N_\epsilon(y)|$ *značí počet objektů v* $N_\epsilon(y)$

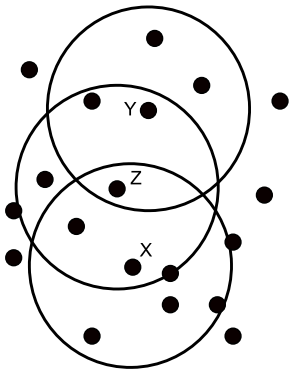
Z této definice 3.2 díky podmínce o počtu objektů vyplývá, že objekty na hranici jsou přímo hustotou dosažitelné z objektů jádra, ale neplatí to naopak. Pouze pokud jsou oba objekty jádra, je jeden přímo hustotou dostupný z druhého a naopak.

Definice 3.3 *Hustotou dosažitelné (Density reachability)*
Mějme dva objekty x a y. O y můžeme mluvit jako o hustotou dosažitelném z x, jestliže existuje posloupnost $x_1, ..., x_n$ objektů, pro které platí $x_1 = x$ a $x_n = y$ a x_{j+1} je přímo hustotou dosažitelné z x_j.



Obrázek 7: Příklad: objekt na hranici přímo hustotou dosažitelný z objektu jádra, ale objekt jádra není přímo hustotou dosažitelný z objektu na hranici

Z této definice 3.3 díky podmínce, že x_{j+1} je přímo hustotou dosažitelné z x_j, vyplývá, že objekty na hranici jsou hustotou dosažitelné z objektů jádra, ale neplatí to naopak. Pokud jsou oba objekty objekty jádra, je jeden přímo hustotou dosažitelný z druhého a naopak. Pro dva objekty na hranici stejného shluku nemusí platit, že by jeden byl hustotou dosažitelný s druhým a naopak.



Obrázek 8: Příklad: objekt x hustotou spojený s objektem y a objekt z, ze kterého jsou objekty x a y hustotou dosažitelné

Definice 3.4 *Hustotou spojený (Density connected)*
Dva objekty x a y jsou hustotou spojené, jestliže existuje objekt z takový, že oba objekty x a y jsou hustotou dosažitelné ze z.

Příklad objektů, které jsou hustotou spojené, ukazuje obrázek 8. Objekty, které jsou hustotou spojené, jsou objekty x a y a objekt, ze kterého jsou hustotou dosažitelné, je z.

Definice 3.5 *Každý shluk vytvořený tímto algoritmem musí splňovat dvě vlastnosti:*

1. *Všechny objekty uvnitř shluku jsou navzájem hustotou spojené*
2. *Jestliže je objekt hustotou spojený na nějaký objekt uvnitř shluku, pak je součástí tohoto shluku*

3.4.1 Popis DBscan algoritmu

Časová složitost tohoto algoritmu je závislá na způsobu uložení jednotlivých shlukovaných objektů. V případě použití R-stromu nebo KD-stromu lze docílit časové složitosti $O(NlogN)$. Algoritmus DBscan je popsán následujícím algoritmem 4.

Algoritmus 4 : Představuje algoritmus DBscan [10]

function DBSCAN(sada_objektů N, ϵ-sousedství ϵ, minimální_počet_objektů m)
    repeat
        výběr (libovolného) dosud nezpracovaného objektu o
        určení objektů patřících do ϵ objektu o
        if (objektů v ϵ objektu o je více než m) **then**
            je vytvořen shluk
        else
            je objekt o považován za šum, později může být nalezen jakou součást ϵ jiného objektu a umístěn do jiného shluku
        end if
        if (objekt o je určen jako součást shluku) **then**
            ϵ-sousedství objektu o je také určeno jako součást shluku
            všechny objekty, které jsou nalezeny uvnitř ϵ daného objektu o, jsou vloženy spolu se svým ϵ do shluku (tento proces pokračuje dokud není odhalen celý shluk)
        end if
    until (Dokud nejsou zpracovány všechny objekty) ▷ Tímto jsou buď všechny objekty součástí některého shluku nebo jsou rozpoznány jako šum
end function

Ač tento algoritmus dosahuje dobrých výsledků (umí rozpoznat shluky libovolného tvaru), má také znatelnou nevýhodu. Tou je právě volba parametrů e (velikost ϵ-sousedství) a m (minimální počet objeků pro vytvoření shluku). Tyto parametry nastavuje uživatel algoritmu a pro jejich určení neexistuje jiná metodika, než je stanovit na základě experimentů s konkrétními daty.

3.4.2 Možnosti optimalizace algoritmu

Optimalizace tohoto algoritmu pro řídká data spočívala v úpravě jednotlivých funkcí pro výpočet míry nepodobnosti dvou objektů tak, aby nemuselo docházet k procházení všech atributů jednotlivých objektů, ale pouze těch nenulových. Z tohoto důvodu se jeví jako nejlepší použití Kosinovy míry. Pro zefektivnění tohoto algoritmu se jeví jako velmi výhodé naimplementovat databázi shluovaných objektů za použití R-stromu (nebo jeho varianty) nebo KD-stromu. Implementace vytvořená v rámci této práce používá sekvenční databázi, která tento algoritmus značně zpomaluje.

3.5 OPTICS

OPTICS [2] je velmi podobný algoritmu DBscan, ale zaměřuje se na jeho hlavní slabinu, a tou je problému identifikace smysluplných shluků v datech s proměnnou hustotou. Ve svém principu je funkce algoritmu stejná jako u algoritmu DBscan pro nekonečný počet parametrů ϵ_i menších než ϵ - algoritmu zadaná generující vzdálenost. Hlavním rozdílem je, že OPTICS nepřiděluje objektům členství ve shluku. Místo toho ukládá pořadí, ve kterém jsou objekty zpracovány a dále dvě informace, na jejichž základě by DBscan rozhodl členství ve shluku. Tyto dvě informace jsou: **core-distance a reachability-distance**. V případě použití efektivní databáze pro uložení objektů je časová složitost tohoto algoritmu stejná jako u algoritmu DBscan, tedy $O(NlogN)$.

Definice 3.6 *Core-distance objektu p.*
Nechť p je objekt z databáze D, ϵ je hodnota vzdálenosti, $N_\epsilon(p)$ je ϵ-sousedství objektu p, MinPts je přirozené číslo a MinPts-distance(p) je vzdálenost objektu p k jeho MinPts-tému objektu. Pak, vzdálenost core-distance objektu p je definována následovně:

$$MinPts(p) = \begin{cases} UNDEFINED, pro|N_\epsilon(p)| \prec MinPts \\ MinPts - distance(p), jinak \end{cases}$$

Definice 3.7 *Reachability-distance objektu p s respektem na objekt o.*
Nechť p a o je objekt z databáze D, $N_\epsilon(o)$ je ϵ-sousedství objektu o, MinPts je přirozené číslo. Pak, vzdálenost reachability-distance objektu p s respektem na o je definována následovně:

$$MinPts(p,o) = \begin{cases} UNDEFINED, pro|N_\epsilon(o)| \prec MinPts \\ max(core - distance(o), distance(o,p)), jinak \end{cases}$$

Stejně jako algoritmus DBscan vyžaduje OPTICS dva parametry: ϵ - určující nejmenší hustotu, kterou chceme rozpoznat a MinPts značící minimální počet objektů pro vytvoření nového shluku. Pro hlubší popis tohoto algoritmu doporučujeme publikaci [2].

3.5.1 Možnosti optimalizace algoritmu

Optimalizace tohoto algoritmu pro řídká data byla, díky podobnosti s algoritmem DBscan, v podstatě totožná jako u algoritmu DBscan. Provedli jsme úpravu jednotlivých funkcí pro výpočet míry nepodobnosti dvou objektů tak, aby nemuselo docházet k procházení všech atributů jednotlivých objektů, ale pouze těch nenulových. Z tohoto důvodu se jeví jako nejlepší použití Kosinovy míry. Pro zefektivnění tohoto algoritmu se jeví jako velmi výhodé naimplementovat databázi shlukovaných objektů za použití R-stromu (nebo jeho varianty) nebo KD-stromu. Implementace vytvořená v rámci této práce používá databázi objektů, která neumožňuje efektivní vyhledání nejbližších sousedů a tím tento algoritmus značně zpomaluje.

4 Datové struktury pro využití při shlukování

V této kapitole se zaměříme na popis datových struktur pro zefektivnění algoritmů DBscan a OPTICS. Těmito strukturami jsou například R-strom (a jeho další varianty) nebo KD-strom. První jmenovaný nebyl v rámci této práce naimplementovaný, je zde uveden pouze jako jedna z možností, jak lze algoritmy DBscan a OPTICS zoptimalizovat. Datová struktura KD-strom byla také naimplementována, ale nebyla zatím pro optimalizaci algoritmu DBscan použita.

4.1 R-strom

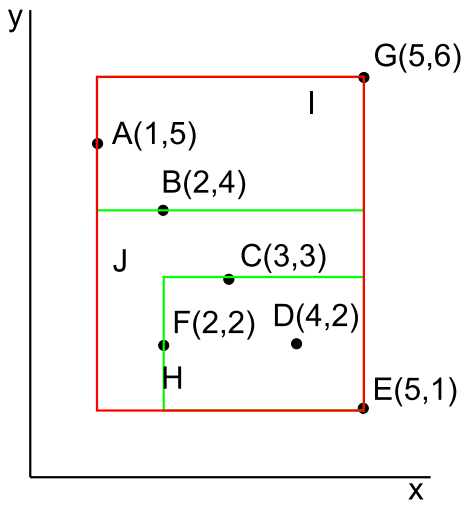
Základní verze struktury R-strom byla poprvé popsána A. Gutmannem [12]. Je jednou z nejznámějších vícerozměrných datových struktur. Struktura R-strom je hierarchická datová struktura podobná datové struktuře B-strom [3]. Objekty v n-rozměrném prostoru jsou shlukovány do minimálních ohraničujících obdélníků (MOO). Každý uzel R-stromu odpovídá MOO, který ohraničuje jeho potomky. Listy R-stromu obsahují ukazatele na databázové objekty, místo aby obsahovaly ukazatele na podřízené uzly. Jednotlivé uzly R-stromu se implementují jako diskové stránky. R-strom podporuje bodové a rozsahové dotazy a dotazy na k-nejbližších sousedů. R-strom může obsahovat MOO, které se překrývají.

Definice 4.1 *Datová struktura R-strom musí splňovat následující podmínky:*

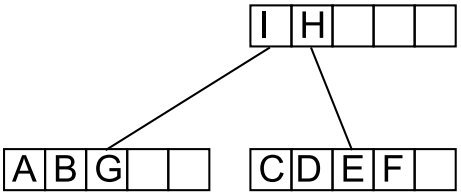
- *Každý listový uzel (pokud se nejedná o kořenový uzel) může obsahovat maximálně M záznamů a minimální povolený počet záznamů je $m \leq \frac{M}{2}$. Každý záznam listového uzlu je ve tvaru (moo, oid), kde moo je MOO obsahující objekt a oid je identifikátor objektu.*
- *Počet záznamů v každém vnitřním uzlu je opět maximálně M a nejméně $m \leq \frac{M}{2}$. Každý záznam vnitřního uzlu je ve tvaru (moo, p), kde p je ukazatel na podřízený uzel a moo je MOO, obsahující minimální ohraničující obdélníky svých podřízených uzlů.*
- *Minimální povolený počet záznamů v kořenovém uzlu je 2. V případě, že je listovým uzlem, může obsahovat 0 záznamů nebo 1 záznam.*
- *Všechny listové uzly jsou na stejné úrovni.*

Tato definice byla přejata z [18].

Z definice R-stromu 4.1 vyplývá, že se jedná o vyvážený strom. Na obrázku 9 jsou ukázány objekty A (1,5), B (2,4), C (3,3), D (4,2), E (5,1), F (2,2), G(5,6) a dále jednotlivé MOO, které tyto objekty obsahují: H, I a J. Na obrázku 10 je ukázán R-strom odpovídající obrázku 9, kdy $M = 5$ a $m = 3$. Je patrné, že pro obrázek 9 může existovat i jiný R-strom. Výsledný R-strom je závislý na pořadí vkládání záznamů do R-stromu nebo na mazání záznamů v R-stromu.



Obrázek 9: Příklad: objekty a jednotlivé MOO ve dvoudimenzionálním prostoru



Obrázek 10: Příklad: R-strom odpovídající obrázku 9

Popis jednotlivých algoritmů použitých pro vkládání, mazání, bodový dotaz, rozsahový dotaz, spojování uzlů nebo rozdělování uzlů je uveden v poublikacích [12] a [18]. Jednotlivé algoritmy zde nejsou popsány, protože je zde tato datová struktura uvedena pouze jako příklad struktury, která může posloužit optimalizaci shlukovacích algoritmů. Počet načítaných uzlů při operaci vkládání, mazání a bodového dotazu muže být vetší než $logn$. Složitost operace rozsahového dotazu je $O(r*logn)$, kde n je počet indexovaných objektů a r je počet prohledávaných listových uzlů.

4.2 KD-strom

KD-strom je datová struktura pro ukládání k-dimenzionálních dat a je speciálním případem BSP-stromu[9]. Hlavní výhodou této datové struktury je, že je schopna vykonat efektivně mnoho různých typů dotazů.

KD-strom je binární strom, který rekurzivně dělí na časti celou množinu objektů. Obsah každého uzlu je dán tabulkou 3.

Název	Popis
med	objekt, určený mediánem atributu, který je dán hodnotou položky split
split	dimenze, podle které se provádí rozdělení
left	levý podstrom daného uzlu - vznikl při dělení dimenze
right	pravý podstrom daného uzlu - vznikl při dělení dimenze

Tabulka 3: Položky uzlů KD-stromu [20]

Položka **med** reprezentuje objekt, který je mediánem podle atributu určeného hodnotou **split** vzorových objektů. Atribut **med** je indexem uzlu a dělí prostor na dva podprostory v závislosti na rozdělení nadroviny uzlu. Všechny objekty v levém podprostoru jsou reprezentovány položkou **left** (reprezentující levý podstrom uzlu) a objekty v pravém podprostoru jsou reprezentovány položkou **right** (reprezentující pravý podstrom uzlu). Rozdělovaná nadrovina prochází skrz **med** a je kolmá na směr zadaný položkou **split**. Jestliže **i** bude hodnota položky **split**, bude objekt nalevo od **med**, pokud bude hodnota i-tého atributu objektu menší než i-tá hodnota atributu položky **med**. Podobně je definováno, kdy objekt patří do pravého podstromu. Pokud nemá uzel žádného potomka, potom není vyžadováno rozdělení nadroviny.

4.2.1 Tvorba KD-stromu

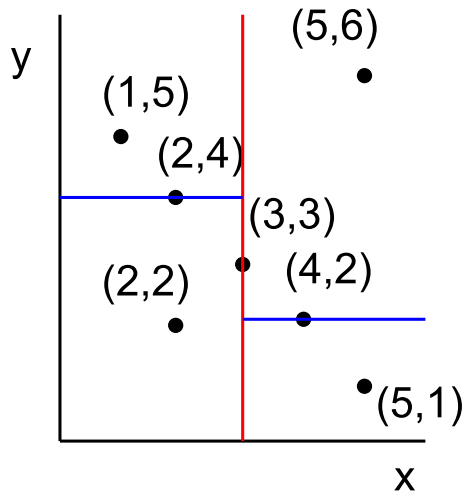
Existuje několik různých způsobů, jak rozdělit datový prostor, a tedy několik různých způsobů. jak vytvořit KD-strom. Jedna z možností, jak provést dělení datového prostoru, vypadá následovně:

- Každý uzel stromu je spojen s jednou osou, podle které se provede rozdělení datového prostoru.
- Objekty jsou vkládány do pravého (resp. levého) podstromu na základě objektu představujícího medián objektů. Tento objekt je vybrán na základě osy, podle které je prováděno rozdělení datového prostoru.

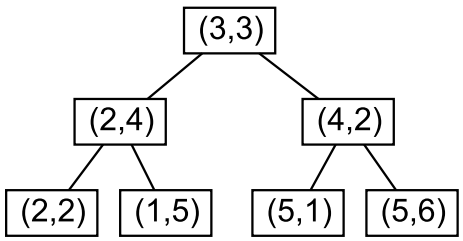
S využitím tohoto přístupu lze popsat algoritmus pro vytvoření datové struktury KD-strom způsobem popsaným algoritmem 5.

Algoritmus 5 : Algoritmus KD-stromu [20]

```
1: function KDTREE(sada_objektů N, hloubka h)
2:     if (N je prázdná) then
3:         Return prázdný strom
4:     else
5:         split = h mod počet_dimenzí ▷ Určení, podle které osy bude provedeno dělení
6:         seřazení objektů v N podle atributu určeného hodnotou split
7:         median := nalezeni mediánu v N podle hodnoty atributu určeného hodnotou
   split
8:         left := KDTREE(objekty v N menších než median, h + 1)
9:         right := KDTREE(objekty v N vetších než median, h + 1)
10:        Return tento uzel
11:    end if
12: end function
```



Obrázek 11: Příklad: rozložení objektů o dvou dimenzích a způsob, jakým je rozdělena rovina x a y [20]



Obrázek 12: Příklad: KD-strom odpovídající předchozímu obrázku 11 [20]

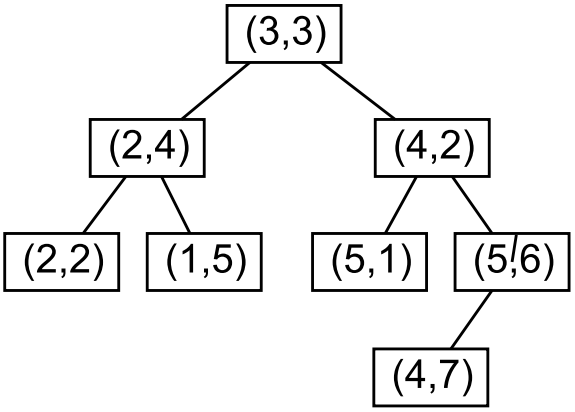
Existují i jiné přístupy k rozdělení uzlu [25], například:

- přístup, kdy je uzel rozdělen tak, aby součet druhých mocnin vzdáleností jednotlivých objektů od středů obou rozdělených stran byl minimální,
- přístup, kdy se uzel dělí na základě mediánové hodnoty atributu, který má největší rozsah hodnot,
- přístup, kdy se uzel dělí na základě střední hodnoty atributu, který má největší rozsah hodnot,
- přístup, kdy se uzel dělí na základě střední hodnoty atributu, který má největší rozsah hodnot, ale v případě, že je po rozdělení jedna strana prázdná, provede se rozdělení podle jiné dimenze.

4.2.2 Inkrementální vkládání objektů do KD-stromu

Přidání nového objektu do KD-stromu je podobné přidávání nového objektu do vyhledávacího binárního stromu. Nejprve sestupujeme stromem z kořenového uzlu až k listovému uzlu. Zda pokračovat pravým nebo levým podstromem určíme na základě toho, zda je příslušná hodnota atributu (určeno podle hloubky uzlu, ve kterém se nacházíme) objektu vetší nebo menší než mediánová hodnota uložena v uzlu. V případě, že jsme dosáhli listového uzlu, bude vytvořen nový uzel a přidán do příslušného podstromu. Do kterého podstromu bude uzel přidán je určeno stejným způsobem, podle kterého jsme určovali jakým podstromem sestoupit stromem. Přídání objektu tímto způsobem způsobí, že se strom stane nevyváženým, což bude mít za následek snížení výkonnosti stromu. Jak moc bude výkonnost snížena závisí na umístění nově přidávaných objektů a na počtu nově přidaných objektů v poměru k velikosti stromu. V případě, že se strom stane příliš nevyvýžený, je třeba jej vyvážit. Časová složitost tohoto inkrementálního vkládání je $O(logN)$.

Na obrázku 13 je uveden příklad, jak bude vypadat KD-strom z obrázku 12 po vložení objektu (4, 7). Tento objekt postupně sestupuje stromem a na základě i-tého atributu (kde i je dáno jako zbytek po dělení úrovně kde se nacházíme počtem atributů objektů) se rozhodne, kterým podstromem daného uzlu se bude pokračovat. Protože v našem případě je první atribut vkládaného objektu větší než první atribut kořenového uzlu, bude se z kořenového uzlu (3, 3) sestupovat pravým podstromem do uzlu (4, 2). Zde se rozhodne podle druhého atributu, že se bude sestupovat opět levým podstromem do uzlu (5, 6). Tento uzel je listem a tak se určí, kterým podstromem by se pokračovalo v sestupování (pravým) a do tohoto podstromu se vkládaný objekt přidá.



Obrázek 13: Příklad: KD-strom po vložení objektu (4, 7) do KD-stromu z obrázku 12

4.2.3 Mazání objektů KD-stromu

Pokud je mazaný objekt listem, je jeho smazání jednoduché. V jiném případě je smazání objektu složité, protože struktura obou podstromů tohoto uzlu je dána objektem, který chceme odstranit. Jedno z řešení by bylo znovu sestavit strom pod smazaným objektem, ale časová náročnost této operace by byla příliš vysoká. Dalším řešením je označení uzlu jako smazaný, ponechat jej ve struktuře, ale nepřihlížet k němu při zpracování dotazů. V okamžiku, kdy je strom znovu sestaven, jsou všechny uzly, označené jako smazané, vynechány.

4.2.4 Operace bodového dotazu

Tato operace slouží pro vyhledání konkrétního uloženého objektu. Během prohledávání se postupuje od kořene k listu. Prochází se postupně rovinami obsahujícími hledaný objekt. Časová složitost této operace je $O(h)$, kde h je výška stromu.

4.2.5 Operace rozsahového dotazu

Tato operace slouží pro vyhledání objektů nacházejících se v nadrovině dotazu. Během prohledávání se postupuje od kořene k listu. Prochází se postupně rovinami protínajícími dotazovací nadrovinu. Celý algoritmus rozsahového dotazu je popsán následujícím algoritmem 6.

4.2.6 Operace vyhledání nejbližšího souseda

Tato operace slouží k efektivnímu nalezení nejbližšího objektu k zadanému objektu. Toto efektivní vyhledání je dáno rychlou eliminací velké části prohledávaného prostoru. Algoritmus vyhledání nejbližšího souseda je popsán následujícím algoritmem 7.

Algoritmus 6 : Algoritmus pro rozsahový dotaz KD-stromu [20]

```
1: function ROZSAHOVY_DOTAZ(uzel u, Rozsah r)
2:     if (s je list) then
3:         if (s je v rozsahu r) then
4:             s je nahlášen jako odpovídající dotazu
5:         else
6:             if (levý podstrom s je plně v rozsahu r) then
7:                 levý podstrom s je nahlášen jako odpovídající dotazu
8:             else
9:                 if (levý podstrom s protíná rozsah r) then
10:                    ROZSAHOVY_DOTAZ(levý potomek s, r)
11:                end if
12:            end if
13:            if (pravý podstrom s je plně v rozsahu r) then
14:                pravý podstrom s je nahlášen jako odpovídající dotazu
15:            else
16:                if (pravý podstrom s protíná rozsah r) then
17:                    ROZSAHOVY_DOTAZ(pravý potomek s, r)
18:                end if
19:            end if
20:        end if
21:    end if
22: end function
```

Algoritmus 7 : Algoritmus popisuje dotaz pro nalezení nejbližšího souseda v KD-stromu [20]

```
1: function NNS(Strom s, Uzel u)
2:     algoritmus rekurzivně sestoupí stromem stejným způsobem, jako by se pokusil
       přidat u do s
3:     jakmile je dosaženo uzlu, který je listem, je tento uzel označen jako NN (Nearest
       neighbor)
4:     algoritmus se následně vrací z rekurze a ověří, zda druhá větev (jiná, než přes
       kterou se vrátil) nemůže obsahovat uzel bližší než NN
5:     if (druhá větev může obsahovat blížší uzel než NN) then
6:         je tato větev prohledána a pokračuje se krokem 4
7:     else
8:         pokračuje se krokem 4
9:     end if
10:    vyhledání skončí v okamžiku dosažení kořene stromu
11: end function
```

4.2.7 Další operace KD-stromu

KD-strom podporuje operace jako je vyhledání k nejbližších sousedů. Množství uzlů, které budou při těchto operacích prohledány, je značně ovlivněno rozložením jednotlivých objektů v prostoru. Konkrétní popis těchto algoritmů lze najít v [20].

4.2.8 Volba vhodné datové struktury

Obě výše uvedené datové struktury slouží pro indexování prostorových dat a umožňují provádět podobné operace. Vhodnost použití dané struktury závisí na našich požadavcích. Při výběru datové struktury bychom měli zvážit následující:

- jaké operace požadujeme (vyhledání nejbližšího souseda, atd.)
- množství indexovaných dat (při malém množství dat je datová struktura typu R-strom zbytečná),
- zda budou data na disku nebo v paměti (na disku - R-strom, v paměti - KD-strom),
- frekvenci změn dat (R-strom je samovyvažující a proto je vhodný pro použití s často probíhajícími operacemi vkládní a mazání, KD-strom by bylo nutné jednou za čas celý přebudovat, proto se nehodí pro použití s velkou frekvencí vkládání a mazání).

5 Popis implementace

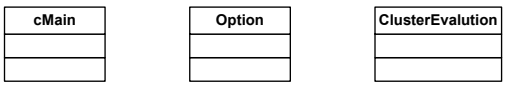
Při implementaci shlukovacích algoritmů, vytvořené v rámci této práce, jsme vycházeli z implementace těchto algoritmů v nástroji Weka[25]. Tuto implementaci jsme převedli z jazyka Java do jazyka C# a poté jsme provedli následující kroky pro úpravu zdrojových kódů.

- Vypustili jsme nepodstatné části kódu.
- Jelikož je Weka[25] napsána tak, že pracuje s jednotlivými typy (třídami, rozhraními, apod.) prostřednictvím reflexe, pro tuto práci byl tento přístup nevyhovující a proto bylo nutné upravit zdrojové kódy, aby reflexi nevyužívaly.
- Jednotlivé shlukovací algoritmy jsme doplniliy o dodatečné schopnosti, jako je například možnost použití různých algoritmů pro výpočet míry nepodobnosti dvou objektů.
- Doimplementovali jsme jednotlivé třídy pro práci s daty (parsery, třídu pro práci s řídkými vektory, apod.).
- Provedli jsme optimalizaci jednotlivých algoritmů, aby využily výhod řídkých dat, protože implementace z nástroje Weka[25] nepředpokádá práci s řídkými daty.

Celá implementace je rozvržena do tří knihoven (Clusters, Util a DataStructure) a jednoho programu, který byl následně použit pro otestování jednotlivých shlukovacích algoritmů. V následujících podkapitolách si tyto knihovny podrobněji popíšeme.

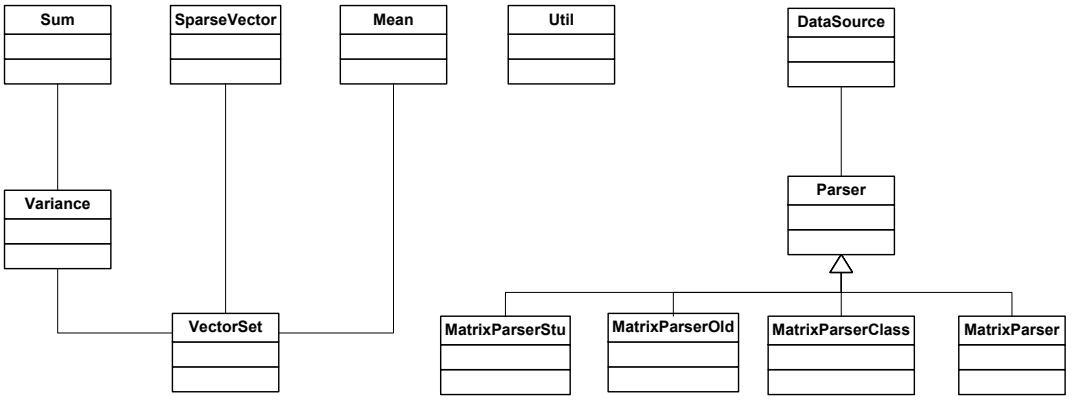
5.1 Popis programu použitého pro experimenty

Pro experimentování s jednotlivými shlukovacími algoritmy jsme vytvořili jednoduchý program využívající jednotlivé níže uvedené knihovny. Na obrázku 14 je znázorněn třídní diagram tohoto programu. Tenoto program je složen ze tří tříd:



Obrázek 14: Třídní diagram popisující program použitý pro testování

- **Option** - třída pro obsluhu parametrů příkazového řádku,
- **ClusterEvalution** - třída, která podle voleb příkazového řádku provede experiment,
- **cMain** - třída sloužící jako vstupní bod aplikace.



Obrázek 15: Třídní diagram knihovny Utils

5.2 Knihovna Utils

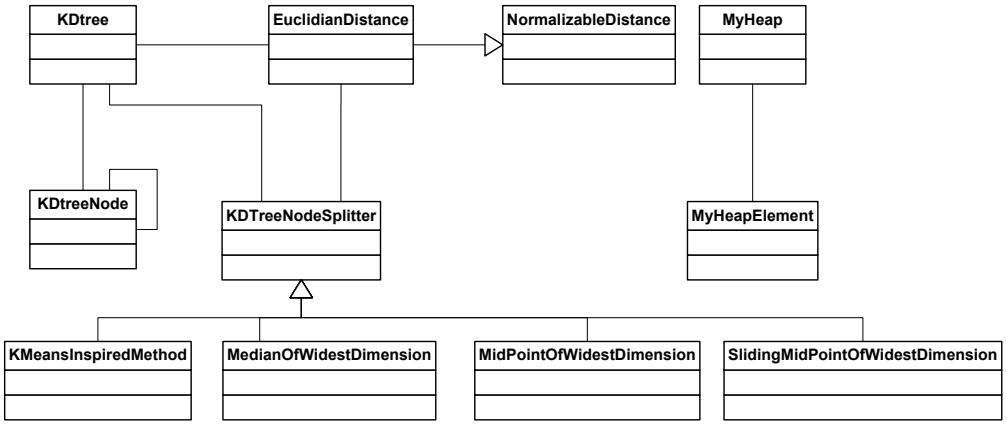
Tato knihovna obsahuje implementaci pomocných tříd nezbytných pro funkci jednotlivých algoritmů.

Na obrázku 15 je třídní diagram knihovny Utils. V tomto diagramu jsme z důvodu úspory prostoru vypustili metody a vlastnosti jednotlivých tříd a diagram zachycuje vztahy mezi třídami pouze této knihovny. Nyní si tento diagram popíšeme:

- **SparseVector** - představuje implementaci řídkého vektoru,
- **VectorSet** - představuje implementaci množiny řídkých vektorů,
- **Mean** - tato třída je použita pro výpočet řídkého vektoru, jehož atributy jsou aritmetickým průměrem atributů vektorů uložených konkrétní instani třídy VectorSet,
- **Variance** - tato třída, je použita pro výpočet řídkého vektoru jeho atributy představují směrodatnou odchylku atributů vektorů uložených v konkrétní instanci třídy VectorSet,
- **Sum** - pomocná třída třídy Variance,
- **Util** - třída obsahující jednoduché metody pro formátování řetězců,
- **Parser** - jedná se o abstraktní bázovou třídu pro jednotlivé parsery souborů s daty,
- **MatrixParserXYZ** - tyto třídy jsou konkrétní implementací parseru pro konkrétní datové soubory,
- **DataSource** - prostřednicvím této třídy jsou, za použití daného parseru, načítány řídké vektory ze souborů.

5.3 Knihovna DataStructure

Tato knihovna obsahuje implementaci KD-stromu a jednotlivých pomocných tříd. Jednotlivé třídy z této knihovny využívají třídy z knihovny Utils. Na obrázku 16 je třídní



Obrázek 16: Třídní diagram knihovny DataStructure

diagram knihovny DataStructure. V tomto diagramu jsme z důvodu úspory prostoru vypustili metody a vlastnosti jednotlivých tříd a tento diagram zachycuje vztahy mezi třídami pouze této knihovny. Nyní si tento diagram popíšeme:

- **KDtree** - tato třída představuje implementaci KD-stromu,
- **KDtreeNode** - tato třída představuje uzel KD-stromu,
- **NormalizableDistance** - bázová třída tříd pro výpočet míry nepodobnosti dvou objektů, umožňuje normalizovat objekty v dané množině objektů a určit rozsahy jednotlivých atributů objektů dané množiny objektů,
- **EuclidianDistance** - je odvozena od NormalizableDistance a umožňuje určit Euklidovskou míru dvou objektů (dle vzorce (7)),
- **KDtreeNodeSplitter** - jedná se o bázovou třídu tříd realizující rozdělení uzlu KD-stromu (odvozené třídy realizují konkrétní algoritmus dělení uzlu),
- **KmeansInspiredMethod** - tato třída realizuje dělení uzlu tak, aby součet druhých mocnin vzdáleností jednotlivých objektů od středů obou rozdělených stran byl minimální,
- **MedianOfWidestDimension** - třída realizující dělení na základě mediánové hodnoty atributu, který má největší rozsah hodnot (pro dané objekty),

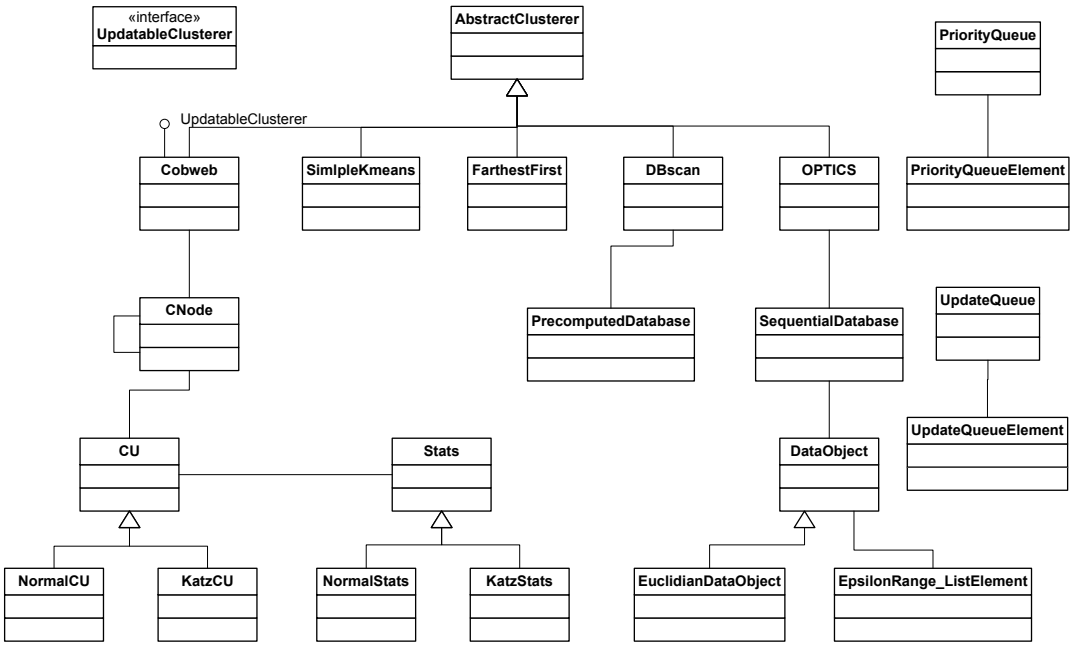
- **MidPointOfWidestDimension** - třída realizující dělení na základě střední hodnoty atributu, který má největší rozsah hodnot (pro dané objekty),
- **SlidingMidPointOfWidestDimension** - třída má podobné chování jako MidPointOfWidestDimension, ale v případě, že je po rozdělení jedna strana prázdná, provede se rozdělení podle jiné dimenze,
- **MyHeap** - pomocná třída, která je použita při implementaci algoritmu vyhledání k nejbližších sousedů,
- **MyHeapElement** - tato pomocná třída představuje prvek MyHeap.

5.4 Knihovna Clusters

Tato knihovna obsahuje implementaci jednotlivých shlukovacích algoritmů, jejichž funkci jsme již dříve popisovali. Jednotlivé třídy z této knihovny využívají třídy z knihovny Utils. Na obrázku 17 je třídní diagram knihovny Utils. V tomto diagramu jsme z důvodu úspory prostoru vypustili metody a vlastnosti jednotlivých tříd a diagram zachycuje vztahy mezi třídami pouze této knihovny. Nyní si tento diagram popíšeme:

- **AbstractClusterer** - jedná se o bázovou třídu jednotlivých shlukovacích algoritmů,
- **UpdatableClusterer** - toto rozhraní slouží jako indikátor toho, že se jedná o inkrementální shlukovací algoritmus,
- **Cobweb** - tato třída, představuje implementaci algoritmu COBWEB (CLASSIT),
- **CNode** - reprezentuje jednotlivé uzly stromu vytvořeného algoritmem COBWEB (CLASSIT),
- **CU** - jedná se o bázovou třídu kategorizačních utilit,
- **XyCU** - jedná se o konkrétní implementaci kategorizační utility,
- **Stat** - třída představuje bázovou třídu pro třídy, které jsou používany při výpočtu kategorizačních utilit,
- **XyStats** - jedná se o konkrétní třídu pro uložení a výpočet statistik používaných při výpočtu kategorizačních utilit,
- **SimpleKmeans** - tato třída reprezentuje implementaci algoritmu K-means,
- **FarthestFirst** - tato třída reprezentuje implementaci algoritmu Farthest First Traversial,
- **DBscan** - tato třída reprezentuje implementaci algoritmu DBscan,

- **PrecomputedDatabase** - tato třída reprezentuje implementaci databáze pro uložení jednotlivých objektů (vektorů) a umožňuje určit všechny sousedy objektu v dané vzdálenosti (určení nejbližších sousedů je vypočteno při vytvoření instance této třídy),
- **OPTICS** - tato třída reprezentuje implementaci algoritmu OPTICS,
- **SequentialDatabase** - tato třída má podobné chování jako třída PrecomputedDatabase až na to, že určení jednotlivých sousedů objektu probíhá až v okamžiku dotázání na tyto sousedy,
- Následující pomocné třídy jsou použity třídou SequentialDatabase při určení sousedů objektů: **DataObject**, **EuclidianDataOBject**, **EpsilonRange_ListElement**, **PriorityQueue**, **UpdateQueueElement**, **UpdateQueue**, **UpdateQueueElement**.



Obrázek 17: Třídní diagram knihovny Clusters

6 Experimenty

V této kapitole jsme se zaměřili na otestování jednotlivých shlukovacích algoritmů a jejich srovnání při zpracování dat, které představují síťový provoz. Experimenty jsme se snažili ověřit vlastnosti jednotlivých shlukovacích algoritmů a vliv nastavení jejich parametrů na účinnost kvalifikace síťového provozu. Dalšími experimenty pak bylo provedeno srovnání účinnosti klasifikace síťového provozu a času učení a testování jednotlivých shlukovacích algoritmů. Všemi testy jsme se snažili o určení algoritmu vhodného pro použití v systémech pro detekci průniku (IDS), což jsou systémy umožňující detekovat pokusy o síťové útoky a průniky podniknuté z vnějších i z interních sítí a reagovat na ně předem definovaným způsobem. Algoritmy, použité při testování, byly Farthest first traversial, K-means a COBWEB/CLASSIT. DBscan nebyl pro experimenty vhodný, protože vlivem použité databáze je nesrovnatelně pomalejší než dříve uvedené algoritmy a OPTICS není pro tento typ experimentů vhodný, protože u jednotlivých objektů neurčí konkrétní příslušnost ke shluku, ale pouze vypočte dvě hodnoty, na jejichž základě o příslušnosti ke shluku rozhodne jiný shlukovací algoritmus.

6.1 Popis dat

Experimenty jsme provedli na skupině dat obsahující pět dvojic souborů: soubor pro učení (5 092 vektorů o 42 atributech) a soubor pro testování (6 890 vektorů o 42 atributech). Každá dvojice představuje data učení a testování pro jeden typ z pěti tříd síťových útoků. Jednotlivé vektory popisující síťový provoz jsou popsány 41 atributy (v rozsahu 0 - 1, tedy není nutné provést normalizaci). Při učení byl používán ještě 42. atribut, určující o jaký síťový provoz se jedná. Při testování je existence tohoto 42. atributu zanedbána a je zjišťováno s jakou přesností je klasifikován daný vektor popisující síťový provoz.

Experimenty probíhaly podle následujícího scénáře:

- byly použity jednotlivé algoritmy pro vytvoření shluků za použití všech vektorů (o plném rozsahu atribů - 42) v souboru pro učení,
- následně bylo určeno, které shluky popisují obyčejný síťový provoz, a které představují síťový útok,
- použitím jednotlivých naučených algoritmů byly klasifikovány vektory z příslušného souboru pro testování a bylo vyhodnoceno procento úspěšnosti (při testování bylo využito pouze prvních 41 atributů jednotlivých vektorů).

6.2 Experimenty s algoritmem Farthest first traversial

Při experimentech s algoritmem Farthest first traversial jsme se snažili odhalit vliv počtu generovaných shluků na úspěšnost klasifikace síťového provozu a na časy učení a testování. Tabulky 4, 5 a 6 uvádí jednotlivé naměřené výsledky. Z nich je možno odvodit, že čas trénování a testování roste s počtem generovaných shluků u všech použitých měr. Z naměřených výsledků nelze přesně určit, při použití které míry bude algoritmus Farthest

first traversial nejrychlejší. U tohoto algoritmu se díky jeho jednoduchosti vliv použité míry neprojeví. Dva nejlepší výsledky (při použití různých měr) dosažené při použití algoritmu Farthest first traversial na detekování třídy útoku Normal jsou zvýrazněny v tabulkách 4, 5 a 6.

Počet generovaných shluků	Čas učení[s]	Čas testování[s]	Úspěšnost[%]
10	2,99	5,4	74,82
20	6,89	6,71	74,73
30	8,42	14,08	**81,86**
40	12,72	22,18	77,9
50	15,21	26,69	77,29
100	25,24	39,91	82,03

Tabulka 4: Výsledky algoritmu FFT při použití Kosinovy míry pro určení třídy útoku Normal

Počet generovaných shluků	Čas učení[s]	Čas testování[s]	Úspěšnost[%]
10	3	4,04	71,96
20	5,96	8	72,82
30	8,75	12,19	80,89
40	11,95	16,45	76,75
50	14,49	19,96	81,1
100	27,62	37,9	84,92

Tabulka 5: Výsledky algoritmu FFT při použití Euklidovy míry pro určení třídy útoku Normal

Počet generovaných shluků	Čas učení[s]	Čas testování[s]	Úspěšnost[%]
10	3,45	4,1	74,76
20	6,52	9,1	74,86
30	9,44	13,13	73,9
40	12,1	16,79	79,77
50	14,92	20,37	75,38
100	28,35	38,87	**85,09**

Tabulka 6: Výsledky algoritmu FFT při použití Manhattenské míry pro určení třídy útoku Normal

6.3 Experimenty s algoritmem K-means

Při experimentech s algoritmem K-means jsme se snažili stejně jako u algoritmu Farthest first traversial, odhalit vliv počtu generovaných shluků na úspěšnost klasifikace síťového provozu a na časy učení a testování. Tabulky 7, 8 a 9 uvádí jednotlivé naměřené výsledky. Z nich je možno odvodit, že čas trénování a testování roste s počtem generovaných shluků u všech použitých měr. Z naměřených výsledků lze odvodit, že použití Kosinovy míry výrazně urychlí časy učení i testování. Dva nejlepší výsledky (při použití různých měr) dosažené při použití algoritmu K-means na detekování třídy útoku Normal jsou zvýrazněny v tabulkách 7, 8 a 9.

Počet generovaných shluků	Čas učení[s]	Čas testování[s]	Úspěšnost[%]
10	48,17	4,71	74,08
20	63,13	12,22	79,27
30	167,41	14,6	86,53
40	217,64	19,77	90,75
50	315,78	19,88	86,69
100	794,6	32,72	**98,64**

Tabulka 7: Výsledky algoritmu K-means při použití Kosinovy míry pro určení třídy útoku Normal

Počet generovaných shluků	Čas učení[s]	Čas testování[s]	Úspěšnost[%]
10	48,97	6,91	74,24
20	104,52	15,33	81,42
30	240,41	22,56	93,41
40	351	19,98	84,16
50	395,82	29,73	**98,16**
100	816,44	38,35	97,88

Tabulka 8: Výsledky algoritmu K-means při použití Euklidovy míry pro určení třídy útoku Normal

Počet generovaných shluků	Čas učení[s]	Čas testování[s]	Úspěšnost[%]
10	48,28	5,64	79,19
20	125,6	12,47	82,19
30	237,76	12,77	82,83
40	489,43	18,27	84,8
50	547,88	21,57	97,65
100	1250,44	48,94	96,46

Tabulka 9: Výsledky algoritmu K-means při použití Manhattenské míry pro určení třídy útoku Normal

6.4 Experimenty s algoritmem COBWEB/CLASSIT

Při experimentech s algoritmem COBWEB/CLASSIT jsme se snažili odhalit závislost parametrů Acuity a Cutoff na generovanou hierarchii shluků a jak tyto parametry ovlivní úspěšnost klasifikace síťového provozu a časy učení a testování.

Tabulka 10 uvádí výsledky experimentů s parametrem Acuity při konstantní hodnotě parametru Cutoff, který je roven hodnotě 0,1 a použité kategorizační utilitě pro atributy s normálním rozdělením. U testovaných dat byla měněna hodnota parametru Acuity v intervalu 0,01 - 0,225 a každý v každém kroku byl tento parametr posunut o 0,025. Z naměřených výsledků je zřejmé, že hierarchie shluků je značně ovlivněna hodnotou parametru Acuity. Jen malá změna tohoto parametru způsobí velkou změnu hierarchie. S rostoucí hodnotou tohoto parametru klesá počet uzlů hierarchie. Výška hierarchie (v tabulce sloupec V.H.) je hodnotou parametru Acuity ovlivněna (to je dáno vlivem na počet uzlů hierarchie), závislost však není jednoznačná. V jakém intervalu nastavovat tuto hodnotu nelze určit univerzálně, protože je závislá na použitých datech. Na čase učení se změna tohoto parametru moc neprojevila. O čase testování lze říci, že čím je hodnota parametru Aucity větší (a tím počet uzlů hierarchie menší), tím je čas testování kratší. Pro náš záměr klasifikace síťového provozu nelze přesně určit, jaká hodnota by se měla použít. Příliš malé i příliš velké hodnoty tohoto parametru vykazují nízkou procentuální úspěšnost.

Čas učení[s]	Čas testování[s]	Úspěšnost[%]	Acuity	V. H.	Počet shluků
248,04	6,44	68,42	0,225	7	37
263,5	9,05	80,42	0,2	11	221
354,87	9,67	80,64	0,175	9	206
313,07	11,22	79,03	0,15	10	307
374,59	11,39	82,42	0,125	10	408
326,89	11,73	76,66	0,1	12	528
329,06	13,76	54,98	0,075	11	672
390,43	17,22	73,56	0,05	12	938
352,86	18,83	74,7	0,025	13	1605
278,14	16,13	46,95	0,01	12	2379

Tabulka 10: Výsledky algoritmu COBWEB/CLASSIT při použití kategorizační utility pro atributy s normálním rozdělením, v závislosti na hodnotě parametru Acuity, při konstantní hodnotě parametru Cutoff 0,1, pro určení třídy útoku Normal

Tabulka 11 uvádí výsledky experimentů s parametrem Cutoff při konstantní hodnotě parametru Acuity, který je roven hodnotě 0,1 a použité kategorizační utilitě pro atributy s normálním rozdělením. Z naměřených výsledků je zřejmé, že hierarchie shluků není ovlivněna hodnotou parametru Cutoff tolik, jako změnou hodnoty parametru Acuity. To je patrné už jen z intervalu, v jakém byl nastavován parametr Cutoff, a že změna v hierarchii uzlů nebyla zdaleka taková, jako při úpravě parametru Acuity. Parametr Cutoff byl upravován v rozsahu 0,1 - 1. S rostoucí hodnotou tohoto parametru klesá počet uzlů

hierarchie. Výška hierarchie (v tabulce V.H.) nebyla hodnotou parametru Cutoff téměř ovlivněna. V jakém intervalu nastavovat tuto hodnotu nelze určit univerzálně, protože je závislá na použitých datech a hodnotě parametru Cutoff. Na čase učení se změna tohoto parametru moc neprojevila. O čase testování lze říci, že čím je hodnota Cutoff větší (a tím počet uzlů hierarchie menší), tím je čas testování kratší. Pro náš záměr klasifikace síťového provozu nelze přesně určit. jaká hodnota by se měla použít. Čím nižší nebo vyšší hodnoty, tím nižší byla procentuální úspěšnost.

Čas učení[s]	Čas testování[s]	Úspěšnost[%]	Cutoff	V. H.	Počet shluků
326,89	11,73	76,66	0,1	12	528
318,83	11,5	78,19	0,2	12	500
317,75	10,14	75,89	0,3	12	434
306,95	11,39	78,62	0,4	11	401
374,47	9,89	79,07	0,5	12	430
301,52	9,26	79,84	0,6	12	359
308,14	9,51	79,33	0,7	12	346
302,84	9,15	78,53	0,8	12	330
313,55	9,45	79	0,9	12	319
306,09	8,74	75,75	1	11	305

Tabulka 11: Výsledky algoritmu COBWEB/CLASSIT při použití kategorizační utility pro atributy s normálním rozdělením, v závislosti na hodnotě parametru Cutoff při konstantní hodnotě parametru Acuity (0,1) pro určení třídy útoku Normal

Pro porovnání jednotlivých kategorizačních utilit pro naše experimenty jsme provedli několik testů s algoritmem COBWEB/CLASSIT s kategorizační utilitou určenou pro atributy mající Katzova rozdělení. Výsledky toho experimentu jsou uvedeny v tabulce 12. Z naměřených výsledků je hned patrné, že hodnoty nastavovaných parametrů Acuity a Cutoff, použité pro nastavení algoritmu používajícího kategorizační utilitu pro normální rozdělení atributů, nebyly pro Katzovo rozdělení vhodné. Proto byla experimentálně zvolena hodnota Cutoff 0,01 a hodnota parametru Acuity se měnila v rozmezí 0,9 - 1,05 o 0,05. Jak je z tabulky 12 patrné, měla tato změna parametru Acuity mnohem menší vliv na generovanou hierarchii. Počet generovaných shluků v hierarchii je při použití této kategorizační utility (v porovnání s kategorizační utilitou pro atributy normálního rozdělení) velice malý. Opět ale můžeme říci, že s rostoucí hodnotou tohoto parametru roste počet shluků v hierarchii i výška této hierarchie. V porovnání s experimentem s kategorizační utilitou pro normální rozdělení bylo při tomto experimentu dosaženo mnohem kratšího času učení i testování. Nelze ale s jistotou říci, zda je to následkem použité kategorizační utility nebo tím, že při tomto experimentu byly generovány mnohem menší hierarchie shluků. Pro srovnání úspěšnosti dosažené při použití této kategorizační utility lze říci, že dosažené výsledky byly mnohem horší než v případě použití kategorizační utility pro atributy normálního rozdělení.

Čas učení[s]	Čas testování[s]	Úspěšnost[%]	Acuity	V. H.	Počet shluků
115,55	3,48	51,47	0,9	5	12
166,1	6,84	79,35	0,95	5	16
72,18	5,05	79,68	1	6	23
60,63	5,6	73,45	1,05	7	31

Tabulka 12: Výsledky algoritmu COBWEB/CLASSIT při použití kategorizační utility pro atributy s Katzovým rozdělením, v závislosti na hodnotě parametru Acuity při konstantní hodnotě parametru Cutoff (0,01) pro určení třídy útoku Normal

6.5 Srovnání výsledků experimentů

Tabulky 13, 14 a 15 zobrazují výsledky testů použití jednotlivých algoritmů na klasifikaci jednotlivých tříd útoků. Jednotlivé experimenty proběhly s algoritmy, jejichž parametry byly experimentálně stanoveny tak, aby se dosáhlo co nejlepších výsledků úspěšnosti klasifikace síťového provozu. Z naměřených výsledků je patrné, že většina nejlepších výsledků byla získána algoritmem K-means, který také vykazoval nejmenší výkyvy v úspěšnosti klasifikace, avšak časy učení tohoto algoritmu byly nejhorší. Největší výkyvy vykazoval nejrychlejší algoritmus Farthest first traversial. Algoritmus COBWEB/CLASSIT umožňoval nejrychlejší testování.

Třída útoku	Čas učení[s]	Čas testování[s]	Úspěšnost[%]
Normal	27,62	37,9	84,92
Probe	41,38	41,7	**98,77**
Dos	34,08	46,53	82,64
U2R	29,55	39,98	95,04
R2L	42,2	39,96	99,27

Tabulka 13: Výsledky algoritmu FFT, pro jednotlivé třídy útoků

Třída útoku	Čas učení[s]	Čas testování[s]	Úspěšnost[%]
Normal	807,17	32,69	**98,64**
Probe	464,42	30,21	97,34
Dos	646,18	34,2	**98,84**
U2R	793,92	36,91	**96,75**
R2L	1068,81	31,93	**99,52**

Tabulka 14: Výsledky algoritmu K-means, pro jednotlivé třídy útoků

Třída útoku	Čas učení[s]	Čas testování[s]	Úspěšnost[%]
Normal	334,72	15,39	83,73
Probe	406,89	10,72	97,79
Dos	310,07	10,23	83,12
U2R	315,33	10,37	93,58
R2l	266,78	8,9	97,92

Tabulka 15: Výsledky algoritmu COBWEB/CLASSIT, pro jednotlivé třídy útoků

Pro srovnání výsledků získaných z experimentů při této práci, uvádíme výsledky experimentů se stejnými daty za použití jiných algoritmů (převzato z [23]). Výsledky byly získány použitím Bayesovské sítě (tabulka 16), klasifikačních a regresních stromů (tabulka 17) a NMF (tabulka 18).

Třída útoků	Čas testování[s]	Úspěšnost[%]
Normal	19,02	99,57
Probe	21,04	99,43
DOS	23,02	99,69
U2R	15,23	64,00
R2L	12,11	99,11

Tabulka 16: Výsledky při použití Bayesovské sítě.[23]

Třída útoku	Čas testování[s]	Úspěšnost[%]
Normal	0,18	99,64
Probe	0,03	97,85
DOS	0,05	99,47
U2R	0,02	48,00
R2L	0,03	90,58

Tabulka 17: Výsledky při použití klasifikačních a regresních stromů [23]

Třída útoku	Čas testování[s]	Úspěšnost[%]
Normal	27,39	77,68
Probe	32,48	89,87
DOS	34,95	78,13
U2R	29,90	97,45
R2L	30,00	98,55

Tabulka 18: Výsledky při použití NMF [23]

V tabulce 19 jsou uvedeny časy učení jednotlivých algoritmů implementovaných v rámci této práce pro jednotlivé třídy útoků. Časy učení algoritmů z článku [23] (Baye-

sovská síť, klasifikační a regresní stromy a NMF) nejsou uvedeny, protože ani v tomto článku nebyly uvedeny. V tabulce 20 jsou uvedeny časy testování za použití jednotlivých algoritmů pro jednotlivé třídy útoků. Čas testování za použití algoritmu COBWEB/CLASSIT byl druhý nejrychlejší a naopak časy testování s algoritmy FFT a K-means byly nejpomalejší. V tabulce 21 jsou uvedeny úspěšnosti jednotlivých algoritmů pro jednotlivé třídy útoků. Průměrná úspěšnost jednotlivých algoritmů je výrazně nejlepší u algoritmu K-means (98,218%). Průměrná úspěšnost ostatních algoritmů se pohybuje v rozmezí od 87,112% - 92,36%.

Třída útoku	FFT	K-means	COBWEB/CLASSIT
Normal	27,62	807,17	334,72
Probe	41,38	464,42	406,89
DOS	34,08	646,18	310,07
U2R	29,55	793,92	315,33
R2L	42,2	1068,81	266,78
Průměr	34,966	756,1	326,758

Tabulka 19: Časy učení (v sekundách) jednotlivých algoritmů pro jednotlivé třídy útoků

Třída útoku	FFT	K-means	COBWEB/ CLASSIT	Bayes. síť	K. a r. stromy	NMF
Normal	37,9	32,69	15,39	19,02	0,18	27,39
Probe	41,7	30,21	10,72	21,04	0,03	32,48
DOS	46,53	34,2	10,23	23,02	0,05	34,95
U2R	39,98	36,91	10,37	15,23	0,02	29,9
R2L	39,96	31,93	8,9	12,11	0,03	30
Průměr	41,214	33,188	11,122	18,084	0,062	30,944

Tabulka 20: Časy testování (v sekundách) při použití jednotlivých algoritmů pro jednotlivé třídy útoků

Třída útoku	FFT	K-means	COBWEB/ CLASSIT	Bayes. síť	K. a r. stromy	NMF
Normal	84,92	98,64	83,73	99,57	99,66	77,68
Probe	98,77	97,34	97,79	99,43	97,85	89,87
DOS	82,64	98,84	83,12	99,69	99,47	78,13
U2R	95,04	96,75	93,58	64	48	97,45
R2L	99,27	99,52	97,92	99,11	90,58	98,55
Průměr	92,128	98,218	91,228	92,36	87,112	88,336

Tabulka 21: Úspěšnost (v procentech) při použití jednotlivých algoritmů pro jednotlivé třídy útoků

7 Závěr

V této diplomové práci jsme se zabývali shlukovacími algoritmy, implementací několika z nich a jejich optimalizací pro velký počet objektů velké dimenze. Během řešení této práce se ukázalo, že ne všechny vybrané algoritmy jsou vhodné pro práci s velkým počtem objektů velké dimenze. Jedním z nich je algoritmus K-means, který i přes upravený výpočet míry nepodobnosti (aby brala v úvahu řídká data) může v nejhorším případě po několika iteracích pracovat s hustými vektory popisujícími jednotlivé shluky. Dalším vylepšením by mohl být použití KD-stromu nebo R-stromu pro uložení centroidů a efektivní vyhledání nejbližšího sousední objektu. Dalším je algoritmus COBWEB/CLASSIT. Podařilo se jej sice upravit, aby při výpočtu kategorizační utility bral v úvahu řídká data, ale právě úprava pro řídká data způsobila nutnost nezbytného ošetřování v přístupu k pomocným strukturám, které tento algoritmus zpomalilo. I přes toto zpomalení je námi optimalizovaný algortimus COBWEB mnohem rychlejší než původní implementace v nástroji Weka. Algoritmy DBscan a OPTICS jsou díky použité databázi objektů značně zpomaleny, a tudíž zatím nejsou pro velký počet objektů velké dimenze použitelné. V případě naimplementování databáze s použitím KD-stromu nebo R-stromu jsou tyto algoritmy pro zpracování velkého počtu objektů velké dimenze vhodné [7, 2]. Jediný algoritmus, v rámci práce implementovaný, který je aktuálně schopný zpracovat velký počet objektů velké dimenze, je Farthest first traversial. Ostatní algoritmy je možné ještě zrychlit využitím paralelizace, či v případě algoritmu DBscan a OPTICS, v optimalizaci databáze (použitím struktury R-strom nebo KD-strom), kterou tyto algoritmy používají. Na nutnost optimalizace databáze jsme narazili během naší práce a i když jsme KD-strom naimplementovali bylo již nad rámec této práce jej začlenit do jednotlivých algoritmů. Proto jsme při experimentech používali algoritmy K-means, Farthest first traversial a COBWEB. Pro použití v systémech IDS jsme z námi provedených experimentů zjistili, že nejvhodnějším algoritmem (algoritmem s největší uspěšností klasifikace provozu) je algoritmus K-means s 98,212% úspěšností klasifikace síťového provozu.

8 Literatura

[1] B. Al-Shboul and S.-H. Myaeng. Initializing k-means using genetic algorithms. 2009.

[2] Ankerst, Mihael, Breunig, M. M., Kriegel, Hans-Peter, and J. Sander. Optics: ordering points to identify the clustering structure. *SIGMOD Rec.*, 28(2):49–60, 1999.

[3] R. Bayer. Binary b-trees for virtual memory. In E. F. Codd and A. L. Dean, editors, *SIGFIDET Workshop*, pages 219–235. ACM, 1971.

[4] P. Berkhin. *Survey of Clustering Data Mining Techniques*. Accrue Software, Inc.

[5] K. Borgwardt. Data mining in bioinformatics.

[6] D. Comaniciu. Image segmentation using clustering with saddle point detection.

[7] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. pages 226–231, 1996.

[8] D. H. Fisher. *Knowledge Acquisition Via Incremental Conceptual Clustering*. Kluwer Academic Publisher, 1987.

[9] H. Fuchs, Z. M. Kedem, and B. F. Naylor. On visible surface generation by a priori tree structures. In *SIGGRAPH '80: Proceedings of the 7th annual conference on Computer graphics and interactive techniques*, pages 124–133, New York, NY, USA, 1980. ACM.

[10] G. Gan, C. Ma, and J. Wu. *Data Clustering Theory, Algorithms and Applications*. ASA-SIAM, 2007.

[11] M. A. Gluck and J. E. Corter. Information, uncertainty, and the utility of categories. In *Proceedings of the Seventh Annual Conference of the Cognitive Science Society*, pages 283–287, Hillsdale, NJ, 1985. Lawrence Earlbaum.

[12] A. Guttman. R-trees: a dynamic index structure for spatial searching. pages 47–57, 1984.

[13] D. Hochbaum and D. Shmoys. A best possible heuristic for the k-center problem. *Math. Oper. Res. 10 (1985) 180-184*.

[14] S. S. Khan and A. Ahmad. Cluster center initialization algorithm for k-means clustering. *Pattern Recognition Letters*, 25(11):1293 – 1302, 2004.

[15] J. L. Kolodner. Reconstructive memory: A computer model. *Cognitive Science*, 7:281–328, 1983.

[16] M. Lebowitz. Concept learning in a rich input domain: Generalization-based memory. 1986.

[17] L. M. Experiments with incremental concept formation: Unimem. *Machine Learning*, 02:103–138(36), September 1987.

[18] Manolopoulos, Yannis, Nanopoulos, Alexandros, Papadopoulos, A. N., and Y. Theodoridis. *R-Trees: Theory and Applications (Advanced Information and Knowledge Processing)*. Springer, 1 edition, September 2005.

[19] R. S. Michalski and R. E. Stepp. Learning from observation: Conceptual clustering. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach*, chapter 11, pages 331–364. Tioga, 1983.

[20] A. Moore. *An introductory tutorial on kd-trees*. Number Technical Report No. 209, Computer Laboratory, University of Cambridge. Pittsburgh, PA, 1991.

[21] P. RNDr. Libor Žák. Shluková analýza i. *AUTOMATIZACE*, 47(3):180 – 182, 2004.

[22] N. Sahoo. Incremental hierarchical clustering of text documents. adviser: Jamie Callan, 2006.

[23] V. Snášel, J. Platoš, P. Krömer, and A. Abraham. Feature deduction and design of intrusion detection systems. 2010.

[24] A. Willium. Clustering algorithm for categorical data. *Math. Oper. Res. 10 (1985) 180-184*, 2006.

[25] I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann Series in Data Management Systems. Morgan Kaufmann, second edition, 2005.

A Uživatelský manuál k programu použitého pro experimenty

Testovací program jsme připravili tak, aby přijímal volby příkazového řádku na jejichž základě provede daný experiment. Volby příkazového řádku jsou následující:

- **-c** - volba pro určení typu použitého shlukovacího algoritmu, možné volby jsou:
  - cobweb - určuje, že bude použit algoritmus COBWEB/CLASSIT
  - farthestFirst - určuje, že bude použit algoritmus Farthest first traversial
  - sipleKmeans - určuje, že bude použit algoritmus K-means
  - DBscan - určuje, že bude použit algoritmus DBscan
  - optics - určuje, že bude použit algoritmus OPTICS
- **-T** - určuje název souboru s daty použitými pro učení,
- **-t** - určuje název souboru s daty použitými pro testování,,
- **-r** - určuje, jaký parser bude použit pro načítání dat ze souboru, aktuální možnosti jsou: 1, 2, 3 a 4, volba 4 je určena pro data použitá pri těchto experimentech, ostatní volby byly použity během implementace pro testování správnosti implementace,
- **-o** - jestliže je vybrán algoritmus COBWEB/CLASSIT určuje tato volba použitou kategorizační utilitu a její možné volby jsou:
  - normalCU - výpočet kategorizační utility založený na vzorci (4)
  - katzCU - výpočet kategorizační utility založený na vzorci (5)

  jestliže je vybrán algoritmus Farthest first traversial nebo K-means určuje tato volba použitou míru nepodobnosti dvou objektů a její možné volby jsou:
  - euclidian - výpočet míry nepodobnosti založený na vzorci (7)
  - manhattan - výpočet míry nepodobnosti založený na vzorci (9)
  - cosine - výpočet míry nepodobnosti založený na vzorci (11)
- **-a** - toto volba je využita pouze algoritmem COBWEB/CLASSIT a představuje hodnotu parametru Acuity,
- **-u** - toto volba je využita pouze algoritmem COBWEB/CLASSIT a představuje hodnotu parametru Cutoff,
- **-n** - toto volba je využita pouze algoritmy K-means a Farthest first traversial a určuje počet shluků ke generování,
- **-i** - toto volba je využita pouze algoritmem K-means a představuje maximální počet iterací algoritmu,

- **-s** - toto volba je využita pouze algoritmem K-means a ovlivňuje počáteční pseudonáhodné rozložení vektorů charakterizujících jednotlivé shluky,
- **-e** - toto volba je využita pouze algoritmy DBscan a OPTICS a určuje velikost ϵ-sousedství,
- **-m** - toto volba je využita pouze algoritmy DBscan a OPTICS a určuje minimální počet objektů (vektorů) pro ustanovení shluku,
- **-d** - tato volba určuje atribut, na který nebude brán při testování ohled,